Signalogic DSP Software

Reference Guide

(Revision 2.01)
September 1999
The one source for all of your DSP needs

Signalogic, Inc.
9617 Wendell @ Skillman
Dallas, TX 75243
Tel: (214) 343-0069
Fax: (214) 343-0163

The software described in this document is furnished under license agreement. The software may be used or copied only under the terms of this license agreement (see section 3.1.1).

Copyright © 1994-1999 Signalogic, Inc. All rights reserved.

No part of this manual may be photocopied or reproduced in any form without prior written consent from Signalogic, Inc.

U.S. Government: If Licensee is acquiring the software on behalf of any unit or agency of the United States Government, the following shall apply:

(a) for units of the Department of Defense:
RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data Clause at DFARS 252.227-7013.

(b) for any other unit or agency:
NOTICE -- Notwithstanding any other lease or license agreement that may pertain to, or accompany the delivery of, the computer software and accompanying documentation, the rights of the Government regarding use, reproduction and disclosure are as set forth in Clause 52.227-19(c)(2) of the FAR.

Contractor/manufacturer is Signalogic, Inc., 9617 Wendell @ Skillman, Dallas, TX 75243.

Hyperception is a registered trademark of Hyperception, Inc.
Signalogic and DSPower are registered trademarks, and Real-Time Convolver is a trademark of Signalogic, Inc.
MATLAB is a registered trademark of The MathWorks, Inc.
National Instruments is a registered trademark, and NIDSP is a trademark of National Instruments, Inc.
Borland is a registered trademark of Borland International, Inc.
Microsoft is a registered trademark, and MS-DOS, Windows, and Visual C are trademarks of Microsoft Corporation.
Analog Devices, AT&T, Motorola, NEC, and Texas Instruments are registered trademarks of their respective holders. ADSP-21xx, WE DSP32x, DSP5600x, DSP9600x, 77220, and TMS320xx are trademarks of their respective holders.
Crystal Semiconductor is a registered trademark of Crystal Semiconductor, Inc.
PC is a trademark of IBM Corporation.
All other trademarks belong to their respective holders.

Signalogic, Inc.
9617 Wendell @ Skillman
Dallas, TX 75243
214-343-0069

Signalogic DSP Software System Guide, June 1999
Design and Implementation: Jeff Brower.
Programming: Jeff Brower , Varsha Shamabhat.
Documentation: Jeff Brower, Varsha Shamabhat, Hanh Nguyen.

Contents

1 Block Reference **1**

Overview and Conventions 1
1.1 Code Blocks 4
1.2 Control Blocks 5
For Loop 5
If Then 7
While Loop 8
1.3 Display Blocks 10
1.4 Filter Blocks 34
1.5 Function Blocks 52
1.5.1 Signal Processing Blocks 52
1.5.2 Math Blocks 78
1.5.3 Signal Generation and Source Blocks 87
1.6 Instrument Blocks 95
1.7 Monitor / Debug Blocks 112
1.8 Utility Blocks 116

2 Function Reference **133**

Overview 133
2.1 DSP Engine Functions 136
2.2 Library Functions 137
2.2.1 Engine Manager Functions 138
2.2.2 Hardware Manager Functions 167
2.2.3 Hardware Library Functions 197
2.2.4 Tool Library Functions 271
2.2.5 Signal Processing Library Functions 302
2.3 DSP Source Code Functions 302

3 DSP/Analog Hardware Reference **317**

Overview 317

3.1 DSP/Analog Hardware Documentation 318
3.2 Board Designator, Driver ID, and Classification Table 319
3.3 Hypersignal DSP Source Code Variable List 323

4 Waveform File Formats 367

Overview 367
4.1 Hypersignal Waveform File Format 368
4.2 Microsoft .WAV Waveform File Format 370
4.3 MATLAB Waveform File Format 373
4.4 Acoustic/Audio Waveform File Formats 373
4.4.1 Techron .TDS File Format 374
4.4.2 Ariel SYSid File Format 374
4.4.3 Micro Technology Unlimited .SF File Format 375

5 Definitions 377

Overview 377
5.1 Structure Definitions 377
5.1.1 HWLISTENTRY Structure 378
5.1.2 CONVERSIONINFO Structure 381
5.1.3 DSPDATARDYMSGINFO Structure 386
5.1.4 ERRORMSGINFO Structure 389
5.2 Constant Definitions 391

6 Messages 393

Overview 393
6.1 Library Messages 394
6.2 DSP Engine Messages 419

7 Error and Status Conditions 437

Overview 437
7.1 DSPower-Block Diagram Error and Status Conditions 441
7.2 DLL Error and Status Conditions 441
7.2.1 General DLL Error and Status Conditions 442
7.2.2 Engine Manager Error and Status Conditions 444
7.2.3 Hardware Manager Error and Status Conditions 447
7.2.4 Hardware Library Error and Status Conditions 449
7.2.5 Signal Processing Library Error and Status Conditions 454
7.3 DSP Engine Error and Status Conditions 454

7.3.1 Hypersignal Error and Status Conditions 454
7.3.2 MATLAB Error and Status Conditions 454

Glossary of Terms 363

Index 365

Glossary of Terms

DRAM

Dynamic random access memory. Typically used as large amount of auxiliary memory; for a DSP/analog board, but sometimes used instead of SRAM if the DSP device has enough onchip SRAM. 4-5 times slower and less expensive than SRAM.

SRAM

Static random access memory. Typically used as main program and data memory for a DSP/analog board. 4-5 times faster and more expensive than DRAM.

VxD

Virtual Device Driver. A type of Windows device driver used to perform low-level system functions or communication.

NDA

Non-disclosure agreement

ISR

Interrupt service routine

EX

Extended

NRT

Non-Real-Time

RT

Real-Time

.mak

Filename extension used for Microsoft Visual C/C++ v1.5 project files.

.ide

Filename extension used for Borland C/C++ v4.5 project files.

.prj

Filename extension used for Borland C/C++ v3.1 project files.

extended memory

System memory located above 1024 kbyte. This is the memory referred to when a system is described as having 4 Mbyte or RAM, or 16 Mbyte of RAM, etc. installed.

expanded memory

Additional, usually large, amount of memory added to a PC system, but accessed through a small (64 kbyte) segment in conventional memory space. No longer as prevalent with advent of systems configured with 4 Mbyte or more of standard (extended) memory.

Index

Error! No index entries found.

1 Block Reference

Overview and Conventions

DSPower blocks have been subdivided into the following categories based on their functions.

Within each category, functions are listed alphabetically. Note that with the online Help version of documentation, you can use the Browse buttons (">>" and "<<") to move forward and backwards between different function descriptions.

The following convention is used in the following sections for block reference.

Text Parameters:

The convention used for text parameters is very simple. The required entry is formatted as bold. The optional parameters are enclosed in square brackets. A special case arises when analog channel inputs are being referred to. The convention used is "**an**" where n represents a number.

For example: *[path]**filename**[ext]* would mean that the filename entry is required, but the path and extension are optional. In the case of analog channels, a0 would be channel 0, a1 would be channel 1, etc.

Numerical Parameters

The convention used for Numerical Parameters is similar. If the entry required is a number, then this field contains "n" followed by the required units for the parameter. This is followed by the possible range. Any special details are mentioned later enclosed in-between /* and */. A typical entry would look like this:

n *<units>; <range allowed>;* /* special cases */

For example: ***n*** *sec; >0*; /* 0 = All */

Block Categories

Code Blocks

Control Blocks

Display Blocks

Filter Blocks

Function Blocks

Instrument Blocks

Monitor/Debug Blocks

Utility Blocks

Block Alphabetic List and Cross Reference

The table below provides a cross-reference for all DSPower-Block Diagram blocks to their corresponding Reference Guide descriptions, which provide input, parameter, and output information and user instructions. Each Reference Guide description also contains a "Related Users Guide Topic" section which lists related areas in the Signalogic DSP Software Users Guide. Users Guide information typically is more general in nature, including more extensive source code examples and explanation of relationship with other blocks and software components.

Function Name	**Page**
2-D Spec (Contour)	366
3-D Spec (Waterfall)	379
Add2	456
A/D Input	476
Bode Plot	385
Convolution	429
Correlation	431
Delay	434
Difference Equations	436
Digital Oscilloscope	387
FIR Bandpass Filter Design	405
FIR Bandstop Filter Design	409
FIR Highpass Filter Design	412
FIR Lowpass Filter Design	414

FFT Generation 436
Frequency Zoom 440
Fs Decimation (Divide) 441
Fs Interpolation (Mult) 443
For Loop 371
Group Delay Display 390
iFFT 444
IIR Bandpass Filter Design 417
IIR Bandstop Filter Design 420
IIR Highpass Filter Design 422
IIR Lowpass Filter Design 425
Magnitude Display 391
Phase Display 394
Pole-Zero Display 396
Sampling Rate Conversion 452
Transfer Function Generation
Waterfall Display
Waveform Display/Edit 402
While Loop 374

1.1 Code Blocks

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace

Comments

Comments go here.

Example

Below is an example of an entry in the Code Block block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_code
Label	Code Block
Data Types	DSP | XY | IMG | MAI
Block Types	CODE
Attributes	0
Defines	
Symbol Filename	code
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

1.2 Control Blocks

Control constructs available in the DSPower system library are listed below. For a complete listing of blocks in alphabetical order, see Block Alphabetic Listing and Cross-Reference, above.

Control Blocks

For Loop

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the For Loop block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_forlp
Label	For Loop
Data Types	DSP | XY | IMG | MAT, 0
Block Types	CTRL
Attributes	LOOP
Defines	
Symbol Filename	for
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	
Function Files	

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

If Then

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the If Then block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

Registration Information

Template Filename	t_ifthen
Label	If Then
Data Types	DSP | XY | IMG | MAT
Block Types	CTRL
Attributes	0
Defines	
Symbol Filename	ifthen
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	
Function Files	

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

While Loop

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the While Loop block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_while
Label	While Loop
Data Types	DSP | XY | IMG | MAT
Block Types	CTRL
Attributes	LOOP
Defines	
Symbol Filename	while
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	
Function Files	

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

.

1.3 Display Blocks

Display functions available in the DSPower system library are listed below. For a complete listing of blocks in alphabetical order, see Block Alphabetic Listing and Cross-Reference, above.

Display Blocks

2-D Spectrograph (Contour)

In a 2-D Spectrogram display, multiple FFT frames are displayed at once, with frequency along the vertical axis, time along the horizontal, and linear or log amplitude encoded either with different colors or gray shades. In this format, you are "looking down" at the frequency domain data - for this reason this display is referred to as a "contour plot."

The 2-D Spectrogram allows single- or dual-trace display of one or two waveforms. In the case of a single waveform, the data can be two-channel. Controls include "current active trace" selection, amount of data shown, vertical amplitude limits (which effectively control the amount of quantized amplitude in linear mode or the amount of quantized dB in log mode that is assigned to each color or gray shade), zoom (and zoom-

out), pan, random-access, target window measurement and readout, units, color and background, and surface topology.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1.
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Parameters		
Max. Dynamic Range	**n** dB	specifies the maximum dynamic range, in dB,
Time Spanned	**n** sec; >=0, 0=All	specifies amount of data to be shown in each segment, in seconds; a zero value causes the entire amount of the largest waveform to be displayed
Color Scheme	**C1/C2/G1/G2**	specifies the color scheme to be used for the display. C1 & C2 = color encoding schemes, G1 = Gray level, G2 =
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2

Comments

If no waveform is specified in the Input 2 field, then dual-trace display is disabled, and the waveform in the Input 1 field is displayed in a single-trace display.

If no filename extension is specified for input waveforms, a .frq extension is assumed.

If Output waveforms are not specified and interactive editing changes are made to the displayed waveform, the input file is overwritten with the changes. In all cases of waveform editing, a .bak

waveform file is created, and in interactive mode, prompts are displayed before changes are saved.

Example

Below is an example of an entry in the 2-D Spec Display block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Max. Dynamic Range	60
Time Spanned	0.0
Color Scheme	C2

See Also

3-D Spec (Waterfall)

Bode Plot

Magnitude Display

Phase Display

Power Spectra Display

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_2dspec
Label	2-D Spectrograph (Contour)
Data Type	DSP
Block Type	DISP
Attributes	DISP | MAGDSP | FRQAXIS | ZAXIS | DAOUT
Defines	SPEC2D
Symbol Filename	spec2D
Help Filename	refguide
Graphical Control Interface Filename	i_file, i_wvfrm, i_xaxis, i_yaxis, i_colors, i_dsphlp
Code Files	c_bdisp, c_adisp

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

3-D Spectrogram (Waterfall)

This display is sometimes referred to as a "landscape" or "spectrogram." In a 3-D Spectrogram display, multiple FFT output frames are displayed at once, with frequency along the horizontal axis, amplitude along the vertical and time depicted as an "apparent," or z-axis going into the screen. Each successive FFT output frame is displayed behind the previous and hidden-line removal may be enabled, giving the overall appearance of a three-dimensional display. For this reason, the 3-D Spectrogram display is often referred to as a form of "waterfall display".

The 3-D Spectrogram allows single- or dual-trace display of one or two waveforms. In the case of a single waveform, the data can be two-channel. Controls include amount of data shown, log or linear amplitude or amplitude-squared display, vertical amplitude limits, zoom (and zoom-out), pan, random-access, target window measurement and readout, units, color and background, grid, angle of apparent incline (ranging from

5° to 90°) and hidden-line removal and surface mesh options. Also, the frequency axis can be specified to be linear or log. In the log mode, the log base, number of cycles to display, and minimum frequency axis value can be selected

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1.
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Parameters		
Display Format	**Log / Lin**	specifies log10 or linear magnitude display
Time Spanned	**n** sec; >=0, 0=All	specifies amount of data to be shown in each segment, in seconds; a zero value causes the entire amount of the largest waveform to be displayed
Apparent Angle	**n** degrees; 5..90	specifies angle of apparent incline of z-dimension (time axis)
Surface	**H / M / N**	specifies hidden line removal option: H = hidden lines removed, M = hidden lines removed and with surface mesh, N = none
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2

Comments

If no waveform is specified in the Input 2 field, then dual-trace display is disabled, and the waveform in the Input 1 field is displayed in a single-trace display.

If no filename extension is specified for input waveforms, a .frq extension is assumed.

If Output waveforms are not specified and interactive editing changes are made to the displayed waveform, the input file is overwritten with the changes. In all cases of waveform editing, a .bak waveform file is created, and in interactive mode, prompts are displayed before changes are saved.

Example

Below is an example of an entry in the 3-D Spec Display block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Display format	Log
Time Spanned	0.0
Apparent Angle	45
Surface	H

Entry Notes

1) In the Input fields, if no path information is entered, the value currently specified in the Waveform Files entry in the Directory Configuration dialog box is used. If no filename extension information is entered; a default extension of .frq is assumed.

2) If only Input 1 is entered, then multitrace display is disabled.

3) Output waveform filenames are created only if not blank, and different from their Input waveform filename counterparts. Otherwise, output waveforms are assumed to be identical to input waveforms.

4) If interactive editing is performed on displayed waveforms, the original waveform files are saved with .bak extensions. Upon closing the display block, a prompt is displayed before waveform editing changes are saved.

See Also

2-D Spec (Contour)

Magnitude Display

Phase Display

Power Spectra Display

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_3dspec
Label	3-D Spectrogram (Waterfall)
Data Type	DSP
Block Type	DISP
Attributes	DISP | MAGDSP | FRQAXIS | ZAXIS
Defines	SPEC3D
Symbol Filename	spec3D
Help Filename	refguide
Graphical Control Interface Filename	i_file, i_wvfrm, i_xaxis, i_yaxis, i_colors, i_dsphlp
Code Files	c_bdisp, c_adisp
Function Files	f_3dspec

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

.WAV File Display

To use .WAV files as input to DSPower, a .WAV extension MUST be indicated in the input fields in the data-entry menus. All time domain functions are active with .WAV files as input, EXCEPT for waveform editing with 8-bit .WAV files. Only 16-bit .WAV files are allowed in waveform editing.

There are limitations to the de facto standard .WAV format found in use with most multimedia programs. One of these is the lack of maximum amplitude value, which causes .WAV files to be displayed and processed in Hypersignal as if they contained full-scale values. Another is lack of frequency or wavelet domain information. These are severe limitations, and it is strongly recommended that you convert your .WAV files into Hypersignal .TIM form using File Acquisition, Difference Equations, or time domain waveform editing, as soon as possible before performing serious signal processing work. With that said, it is still worth noting that .WAV file input has been tested with many Hypersignal functions, such as sampling rate conversion, frequency zoom, and minimum phase calculation, using the .WAV files as direct input (and with Hypersignal .TIM, .FRQ, and .PHZ files as typical output).

Inputs	Entry	Description
Input 1	[path]**filename.wav**	waveform to display on trace

		1
Input 2	[path]**filename.wav**	waveform to display on trace 2

Parameters

Framesize	**n** samples, >=0, 0=All	specifies the framesize needed for display

Outputs

Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2

Comments

If no waveform is specified in the Input 2 field, then dual-trace display is disabled, and the waveform in the Input 1 field is displayed in a single-trace display.

A filename extension is necessary. If no filename extension is specified for input waveforms, a .tim extension is assumed.

If Output waveforms are not specified and interactive editing changes are made to the displayed waveform, the input file is overwritten with the changes. In all cases of waveform editing, a .bak waveform file is created, and in interactive mode, prompts are displayed before changes are saved.

Example

Below is an example of an entry in the .WAV File Display block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Framesize	0

See Also

Waveform Display/Edit

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_
Label	.WAV File Display
Data Type	DSP
Block Type	DISP
Attributes	DISP
Defines	
Symbol Filename	disped
Help Filename	refguide
Graphical Control Interface Filename	
Code Files	
Function Files	

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Bode Plot

Online documentation for this section is unavailable in this version. Please consult DSP Engine documentation for further information.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2

Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2

Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Bode Plot block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_bode
Label	Bode Plot
Data Types	DSP
Block Types	DISP
Attributes	DISP | MULDISP | PHZDSP | MAGDSP | FRQAXIS | FRQZOOM | FRQPAN
Defines	BODEDISP

Symbol Filename	phzdisp
Help Filename	refguide
Graphical Control Interface Filenames	i_file, i_wvfrm, i_xaxis, i_yaxis, i_dspopt, i_dsphlp
Code Files	c_bdisp, c_adisp
Function Files	f_magdsp, f_phzdsp

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Digital Oscilloscope

The Digital Oscilloscope can display single-, dual, or overlaid-trace time domain input from previously stored waveform files or any of the currently active analog channel inputs. Various combinations are possible, including viewing that same analog channel with different filtering options. Operation from the DSP chip point of view is always in real-time up to the maximum sampling rate, but the rate of update of display frames to the video system of the host computer depends on the speed of the host I/O bus and video system. The maximum sampling rate possible depends on the DSP/Acquisition board and DSP chip in use.

FIR and IIR created using the Filter design methods supplied or supplied from outside can be "snapped-in" to each trace independently and applied to analog input data in real-time. Maximum filter sizes and real-time operation with filters are dependent

on the sampling rate and DSP/Acquisition board in use.

Controls include timebase, trigger level, trace, and mode, vertical scaling, position and units.

Inputs	Entry	Description
Input 1	[path]**filename**[ext], **an**	waveform/analog input channel to display on trace 1.
Input 2	[path]**filename**[ext], **an**	waveform/analog input channel to display on trace 2
Parameters		
Wrapped / Unwrapped	**U / W**	specifies required sampling frequency
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2

Comments

If no waveform/analog input channel is specified in the Input 2 field, then dual-trace display is disabled, and the waveform in the Input 1 field is displayed in a single-trace display.

If no filename extension is specified for input waveforms, a .tim extension is assumed.

If Output waveforms are not specified and interactive editing changes are made to the displayed waveform, the input file is overwritten with the changes. In all cases of waveform editing, a .bak waveform file is created, and in interactive mode, prompts are displayed before changes are saved.

Example

Below is an example of an entry in the Digital Oscilloscope block:

Input 1	a0
Input 2	a1
Output 1	
Output 2	
Sampling Frequency	15000

See Also

Digital Tape Recorder

Octave Band Analyzer

Spectrum Analyzer

Stimulus and Response

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_dscope
Label	Digital Oscilloscope
Data Type	DSP
Block Type	INST | DISP
Attributes	DISP
Defines	DSCOPE
Symbol Filename	dscope
Help Filename	refguide
Graphical Control Interface Filename	i_dscope
Code Files	c_ainst
Function Files	f_dscope

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Group Delay Display

Online documentation for this section is unavailable in this version. Please consult DSP Engine documentation for further information.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Group Delay Display block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_gddsp
Label	Group Delay Display
Data Types	DSP
Block Types	DISP
Attributes	DISP | PHZDSP | FRQAXIS | FRQZOOM | FRQPAN
Defines	
Symbol Filename	phzdisp
Help Filename	refguide
Graphical Control Interface Filenames	i_file, i_wvfrm, i_axis, i_yaxis, i_dspopt, i_dsphlp
Code Files	c_bdisp, c_adisp
Function Files	f_phzdsp

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Magnitude Display

This function allows frame-by-frame display and inspection of frequency domain magnitude data. It allows single-, dual-, or overlaid-trace display of one or two frequency domain waveforms.

Controls include log or linear amplitude or amplitude squared display, "current active trace" selection, zoom (and zoom-out), pan, vertical amplitude limits, amplitude scaling, random-access, target window, crosshair cursor measurement and readout, units,

grid, selection of vertical axis units, scaling factor, and maximum value.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1.
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Parameters		
Maximum dB Displayed	**n** dB	specifies the maximum dB value to be displayed,
Minimum dB Displayed	**n** dB	specifies the minimum dB value to be displayed
0 dB Reference	**A/S/C/W**	specifies the 0 dB point for log amplitude displays A = autoscale, S = System reference, C = system ref value offset by the current Spectrum Analyzer calibration, W = uses the system ref value offset plus a stored waveform header calibration offset
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2

Comments

If no waveform is specified in the Input 2 field, then dual-trace display is disabled, and the waveform in the Input 1 field is displayed in a single-trace display.

If no filename extension is specified for input waveforms, a .frq extension is assumed.

If Output waveforms are not specified and interactive editing changes are made to the displayed waveform, the input file is overwritten with the changes. In all cases of waveform editing, a .bak waveform file is created, and in

interactive mode, prompts are displayed before changes are saved.

Example

Below is an example of an entry in the Magnitude Display block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Max. dB Displayed	0
Min dB Displayed	0
0 dB Reference	A

See Also

2-D Spec (Contour)

3-D Spec (Waterfall)

Bode Plot

Phase Display

Power Spectra Display

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_magdsp
Label	Magnitude Display
Data Type	DSP
Block Type	DISP
Attributes	DISP | MAGDSP | FRQAXIS | FRQZOOM | FRQPAN
Defines	
Symbol Filename	magdisp
Help Filename	refguide
Graphical Control Interface Filename	i_file, i_wvfrm, i_xaxis, i_yaxis, i_dspopt, i_dsphlp
Code Files	c_bdisp, c_adisp

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Phase Display

This function allows frame-by-frame display and inspection of frequency domain phase data. It allows single-, dual-, or overlaid-trace display of one or two frequency domain waveforms.

Controls include wrapped phase, unwrapped phase, or group delay display format, "current active trace" selection, zoom (and zoom-out), pan, vertical amplitude limits, amplitude scaling, random-access, target window and crosshair cursor measurement and readout, units, grid, and selection of group delay smoothing factor.

Inputs	**Entry**	**Description**
Input 1	[path]**filename**[ext]	waveform to display on trace 1.
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Parameters		
Wrapped / Unwrapped	**U / W**	specifies phase display format. U = Unwrapped, W = Wrapped
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2

Comments

If no waveform is specified in the Input 2 field, then dual-trace display is disabled, and the waveform in the Input 1 field is displayed in a single-trace display.

If no filename extension is specified for input waveforms, a .phz extension is assumed.

If Output waveforms are not specified and interactive editing changes are made to the displayed waveform, the input file is overwritten with the changes. In all cases of waveform editing, a .bak waveform file is created, and in interactive mode, prompts are displayed before changes are saved.

Example

Below is an example of an entry in the Phase Display block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Wrapped / Unwrapped	U

See Also

Bode Plot

Group Delay Display

Magnitude Display

Pole-Zero Display

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_phzdsp
Label	Phase Display
Data Type	DSP
Block Type	DISP
Attributes	DISP | PHZDSP | FRQAXIS | FRQZOOM | FRQPAN
Defines	
Symbol Filename	phzdisp
Help Filename	refguide
Graphical Control Interface Filename	i_file, i_wvfrm, i_axis, i_yaxis, i_dspopt, i_dsphlp
Code Files	c_bdisp, c_adisp
Function Files	f_phzdsp

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Pole-Zero Display

Pole-zero display allows S-plane and Z-plane of IIR filter transfer functions. Input comes from IIR filter coefficient files, which contain Z-domain poles and zeros; an inverse bilinear transformation is performed to map the filter coefficient file values onto the S-plane.

Inputs	Entry	Description
Input	[path]**filename**[ext]	input IIR filter coefficient file
Outputs		
Output	[path]**filename**[ext]	output waveform

Comments

If no filename extension is specified for input waveforms, a .iir extension is assumed.

If Output waveforms are not specified and interactive editing changes are made to the displayed waveform, the input file is overwritten with the changes. In all cases of waveform editing, a .bak waveform file is created, and in interactive mode, prompts are displayed before changes are saved.

Example

Below is an example of an entry in the Pole-Zero Display block:

```
Input    bessel
Output
```

See Also

IIR Bandpass Filter Design

IIR Bandstop Filter Design

IIR Highpass Filter Design

IIR Lowpass Filter Design

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_pzplot
Label	Pole-Zero Display
Data Type	DSP
Block Type	DISP
Attributes	DISP | IIR
Defines	
Symbol Filename	pzplot
Help Filename	refguide
Graphical Control Interface Filename	i_file, i_dsphlp
Code Files	c_bdisp, c_adisp

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Power Spectra Display

This function allows frame-by-frame display and inspection of generated Power Spectra data. It allows single-, dual-, or overlaid-trace display of one or two waveforms.

Controls include log or linear amplitude or amplitude squared display, "current active trace" selection, zoom (and zoom-out), pan, vertical amplitude limits, amplitude scaling, random-access, target window, crosshair cursor measurement and readout, units, grid, selection of vertical axis units, scaling factor, and maximum value.

This function allows frame-by-frame display and inspection of frequency domain phase data. It allows single-, dual-, or overlaid-trace display of one or two frequency domain waveforms.

Controls include wrapped phase, unwrapped phase, or group delay display format, "current active trace" selection, zoom (and zoom-out), pan, vertical amplitude limits, amplitude scaling, random-access, target window and crosshair cursor measurement and readout, units, grid, and selection of group delay smoothing factor.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1.
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Parameters		
Maximum dB Displayed	**n** dB	specifies the maximum dB value to be used for display
Minimum dB Displayed	**n** dB	specifies the minimum dB value to be used for display
0 dB reference	**A/S/C/W**	specifies the 0 dB point for log amplitude displays A = autoscale, S = System reference, C = system reference value offset by the current Spectrum Analyzer calibration, W = uses the system ref value offset plus a stored waveform header calibration offset
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2

Comments

If no waveform is specified in the Input 2 field, then dual-trace display is disabled, and the waveform in the Input 1 field is displayed in a single-trace display.

If no filename extension is specified for input waveforms, a .pwr extension is assumed.

If Output waveforms are not specified and interactive editing changes are made to the displayed waveform, the input file is overwritten with the changes. In all cases of waveform editing, a .bak waveform file is created, and in interactive mode, prompts are displayed before changes are saved.

Example

Below is an example of an entry in the Power Spectra Display block.

Input 1	speech
Input 2	
Output 1	
Output 2	
Maximum dB Displayed	0
Minimum dB Displayed	0
0 dB Reference	A

See Also

Bode Plot

Magnitude Display

Phase Display

Power Spectra Estimation

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_pwrdsp
Label	Power Spectra Display
Data Type	DSP
Block Type	DISP
Attributes	DISP | PWRDSP | FRQAXIS | FRQZOOM | FRQPAN
Defines	
Symbol Filename	magdisp
Help Filename	refguide
Graphical Control Interface Filename	i_file, i_wvfrm, i_xaxis, i_yaxis, i_dspopt, i_dsphlp
Code Files	c_bdisp, c_adisp
Function Files	f_MagDsp

For more information on Registration Information, see Signalogic DSP Software Users

Guide section , Registration Parameters

Spectrum Analyzer

See Spectrum Analyzer in section 1.6, Instrument Blocks.

Wave Display

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Wave Display block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_wavdsp
Label	Wave Display
Data Types	DSP
Block Types	DISP
Attributes	DISP | TIMDSP | EDIT | GENEXT | TIMPAN | DAOUT
Defines	
Symbol Filename	disped
Help Filename	refguide
Graphical Control Interface Filenames	i_file2, i_edit, i_wvfrm, i_xaxis, i_yaxis, i_dspopt, i_dsphlp
Code Files	c_bdisp, c_adisp
Function Files	f_wde

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Waveform Display/Edit

Waveform Display/Edit function is a general function that allows single-, dual-, or overlaid-trace display and editing of one or two waveforms. In the case of a single waveform, the data can be two-channel or complex. Also, waveforms other than time domain data can be displayed and edited. Both cut-and-paste and free-form "rubber-band" editing modes are available. These modes can be freely switched between in editing

sessions, and are fully enabled at all levels of zoom. Cut-and-Paste operations include delete, copy, insert, and overwrite. Display controls include "current active trace" selection, framesize, zoom (and zoom-out), pan, amplitude scaling, random-access, target window and crosshair cursor measurement and readout, units, color and background, grid, and selection of vertical axis units, scaling factor, and maximum value.

Additionally, segments of displayed waveforms can be written to file or sent to the D/A output of the currently selected DSP/Acquisition hardware. A built-in print function is also available that allows screen dumps or publication quality EMS memory regenerations to nearly 200 different printers.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1.
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Parameters		
Framesize	**n** samples, >=0, 0=All	specifies the framesize needed for display
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2

Comments

If no waveform is specified in the Input 2 field, then dual-trace display is disabled, and the waveform in the Input 1 field is displayed in a single-trace display.

If no filename extension is specified for input waveforms, a .tim extension is assumed.

If Output waveforms are not specified and interactive editing changes are made to the displayed waveform, the input file is overwritten with the changes. In all cases of waveform editing, a .bak waveform file is created, and in interactive mode, prompts are displayed before changes are saved.

Example

Below is an example of an entry in the Waveform Display block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Framesize	0

See Also

.WAV Fle Display

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_wde
Label	Waveform Display / Edit
Data Type	DSP
Block Type	DISP
Attributes	DISP | TIMDSP | EDIT | GENEXT | TIMPAN | DAOUT
Defines	
Symbol Filename	disped
Help Filename	refguide

Graphical Control Interface Filename	i_file2, i_edit, i_wvfrm, i_xaxis, i_yaxis, i_dspopt, i_dsphlp
Code Files	c_bdisp, c_adisp
Function Files	f_wde

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

1.4 Filter Blocks

Filter Design functions available in the DSPower system library are listed below. For a complete listing of blocks in alphabetical order, see Block Alphabetic Listing and Cross-Reference, above.

Filter Blocks

Convolution

See Convolution in section 1.5.1, Signal Processing Blocks.

FIR Bandpass Filter Design

This block allows construction of FIR Bandpass filters using the Kaiser Window and Parks-McClellan design methods.

Arbitrary designs, such as multiband and irregular or log frequency passbands, can be specified and realized using the Parks-McClellan method. Filter name, sampling frequency,

bandwidth, transition bandwidths, stopband attenuation, and passband ripple are selectable parameters. Filter length is estimated by the software, but may be controlled explicitly. All designs are first displayed and can be measured in a graphical filter display prior to design. Arbitrary designs can be further edited directly using the mouse. Resulting frequency responses can be compared in the graphical display to original specifications and "tolerance regions".

The software creates a time domain impulse response waveform as output. Also, the software creates both an ASCII "filter coefficient file" and an ASCII "design goal file", which contains the filter specifications and arbitrary filter design, if any. Current maximum filter sizes are 4096 for Kaiser Window design and 768 for Parks-McClellan design.

Filter designs are always displayed graphically prior to actual construction. Several measurement functions are available, which apply to both classical and arbitrary designs. Several editing functions are also available, which apply only to arbitrary designs Measurement functions include target window and crosshair measurement and readout, grid, log/linear frequency and amplitude axes, overlay of actual and desired frequency responses, vertical scaling, and tolerance region display toggle.

Editing functions include cut-and-paste and free-form rubberband modes, control over frequency point-density and "defined" and "undefined" regions, and initialization of response from specifications or "design goal files"

Inputs	Entry	Description
Input Design File	[path]**filename**[ext]	filter design goal file
Parameters		
Design Method	**KW/PM**	specifies the design method to be used. KW = Kaiser Window, PM = Parks-McClellan
Sampling Frequency	**n** Hz; >0	specifies sampling frequency in Hz
Center Frequency	**n** Hz, >0	Specifies the center frequency
Bandwidth	**n** Hz, >0	specifies the bandwidth required in Hz
Left Transition Band	**n** Hz, >0	specifies the left transition band in Hz
Right Transition Band	**n** Hz, >0	specifies the right transition band in Hz
Stopband Attenuation	**n** dB, 1..100	specifies stopband attenuation in dB
Maximum Ripple	**n** dB	specifies maximum passband ripple in dB
Outputs		
Output Filter Coefficient File	[path]**filename**[ext]	output filter coefficient file

Comments

The minimum of the transition bandwidth fields is used to determine the filter order or length. Since classical designs always produce symmetrical filters, the minimum transition bandwidth will control both transition bandwidths. Arbitrary designs can be used to achieve asymmetrical transition bandwidths.

If no extension is specified for the Input filter design filename, a .ffd extension is assumed. The output filter coefficient filename has a default extension of .fir.

Example

Below is an example of an entry in the FIR Bandpass Filter Design block:

Input Design File	parksm
Design Method	PM
Sampling Frequency	48000
Center Frequency	12000
Bandwidth	4000
Left Transition band	500
Right Transition Band	500
Stopband Attenuation	40
Maximum Ripple	1
Output	

See Also

FIR Bandstop Filter Design

FIR Highpass Filter Design

FIR Lowpass Filter Design

IIR Bandpass Filter Design

IIR Bandstop Filter Design

IIR Highpass Filter Design

IIR Lowpass Filter Design

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_firbp
Label	FIR Bandpass Filter Design

Data Type	DSP
Block Type	FILT
Attributes	DISP | FIR
Defines	
Symbol Filename	bandpass
Help Filename	refguide
Graphical Control Interface Filename	i_filtfb, i_filt
Code Files	c_bfilt, c_afilt
Function Files	f_filter, f_fft

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

FIR Bandstop Filter Design

This block allows construction of FIR Highpass filters using the Kaiser Window and Parks-McClellan design methods. See FIR Bandpass Filter Design for more information about FIR Filter design capabilities of the software.

Inputs	**Entry**	**Description**
Input Design File	[path]**filename**[ext]	filter design goal file
Parameters		
Design Method	**KW/PM**	specifies the design method to be used. KW = Kaiser Window, PM = Parks-McClellan
Sampling Frequency	**n** Hz; >0	specifies sampling frequency in Hz
Center Frequency	**n** Hz	specifies the center frequency
Bandwidth	**n** Hz, >0,	specifies the bandwidth in Hz
Left Transition Band	**n** Hz, >0	specifies the left transition band in Hz
Right Transition Band	**n** Hz, >0	specifies the right transition band in Hz

Stopband Attenuation	**n** dB, 1..100	specifies stopband attenuation in dB
Maximum Ripple	**n** dB	specifies maximum passband ripple in dB

Outputs

Output Filter Coefficient File	[path]**filename**[ext]	output filter coefficient file

Comments

The bandwidth field is interpreted as the width of the filter's stopband. Otherwise, there is no difference between bandstop and bandpass. Use a negative value for stopband attenuation.

For FIR bandstop filters, the software automatically adjusts length estimates to be odd values. This is because FIR filters with even symmetry that have a high magnitude value at the Nyquist frequency are required to have an odd length value (which produces the appropriate Nyquist frequency response about n = 0). If you override the length estimate, please continue to use an odd value for bandstop filters.

If no extension is specified for the Input filter design filename, a .ffd extension is assumed. The output filter coefficient filename has a default extension of .fir.

Example

Below is an example of an entry in the FIR Bandstop Filter Design block:

Input Design File	firhp
Design Method	KW
Sampling Frequency	48000
Center Frequency	12000

Bandwidth	4000
Left Transition Band	500
Right Transition Band	500
Stopband Attenuation	40
Maximum Ripple	1
Output	

See Also

FIR Bandpass Filter Design

FIR Highpass Filter Design

FIR Lowpass Filter Design

IIR Bandpass Filter Design

IIR Bandstop Filter Design

IIR Highpass Filter Design

IIR Lowpass Filter Design

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_firbs
Label	FIR Bandstop Filter Design
Data Type	DSP
Block Type	FILT
Attributes	DISP | FIR
Defines	
Symbol Filename	bandstop
Help Filename	refguide
Graphical Control Interface Filename	i_filtfb, i_filt
Code Files	c_bfilt, c_afilt
Function Files	f_filter, f_fft

For more information on Registration Information, see Signalogic DSP Software Users

Guide section , Registration Parameters

FIR Highpass Filter Design

This block allows construction of FIR Highpass filters using the Kaiser Window and Parks-McClellan design methods. See FIR Bandpass Filter Design for more information about filter design capabilities of the software.

Inputs	Entry	Description
Input Design File	[path]**filename**[ext]	filter design goal file
Parameters		
Design Method	**KW/PM**	specifies the design method to be used. KW = Kaiser Window, PM = Parks-McClellan
Sampling Frequency	**n** Hz; >0	specifies sampling frequency in Hz
Bandwidth	**n** Hz, >0,	specifies bandwidth in Hz
Transition Band	**n** Hz, >0	specifies transition band in Hz
Stopband Attenuation	**n** dB, 1..100	specifies stopband attenuation in dB
Maximum Ripple	**n** dB	specifies maximum passband ripple in dB
Outputs		
Output Filter Coefficient File	[path]**filename**[ext]	output filter coefficient file

Comments

For FIR highpass filters, the software automatically adjusts length estimates to be odd values. This is because FIR filters with even symmetry that have a high magnitude value at the Nyquist frequency are required to have an odd length value (which produces the appropriate Nyquist frequency response about n = 0). If you

override the length estimate, please continue to use an odd value for highpass filters

If no extension is specified for the Input filter design filename, a .ffd extension is assumed. The output filter coefficient filename has a default extension of .fir.

Example

Below is an example of an entry in the FIR Highpass Filter Design block:

Input Design File	firhp
Design Method	KW
Sampling Frequency	48000
Bandwidth	4000
Transition Band	500
Stopband Attenuation	40
Maximum Ripple	1
Output	

See Also

FIR Bandpass Filter Design

FIR Bandstop Filter Design

FIR Lowpass Filter Design

IIR Bandpass Filter Design

IIR Bandstop Filter Design

IIR Highpass Filter Design

IIR Lowpass Filter Design

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_firhp

Label	FIR Highpass Filter Design
Data Type	DSP
Block Type	FILT
Attributes	DISP | FIR
Defines	
Symbol Filename	highpass
Help Filename	refguide
Graphical Control Interface Filename	i_filthb, i_filt
Code Files	c_bfilt, c_afilt
Function Files	f_filter, f_fft

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

FIR Lowpass Filter Design

This block allows construction of FIR Lowpass filters using the Kaiser Window and Parks-McClellan design methods. See FIR Bandpass Filter Design for more information about filter design capabilities of the software.

Inputs	**Entry**	**Description**
Input Design File	[path]**filename**[ext]	filter design goal file
Parameters		
Design Method	**KW/PM**	specifies the design method to be used. KW = Kaiser Window, PM = Parks-McClellan
Sampling Frequency	**n** Hz; >0	specifies sampling frequency in Hz
Bandwidth	**n** Hz, >0,	specifies bandwidth in Hz
Transition Band	**n** Hz, >0	specifies transition band in Hz
Stopband Attenuation	**n** dB, 1..100	specifies stopband attenuation in dB
Maximum Ripple	**n** dB	specifies maximum required passband ripple in dB

Outputs		
Output Filter Coefficient File	[path]**filename**[ext]	output filter coefficient file

Comments

For FIR highpass filters, the software automatically adjusts length estimates to be odd values. This is because FIR filters with even symmetry that have a high magnitude value at the Nyquist frequency are required to have an odd length value (which produces the appropriate Nyquist frequency response about n = 0). If you override the length estimate, please continue to use an odd value for highpass filters

If no extension is specified for the Input filter design filename, a .ffd extension is assumed. The output filter coefficient filename has a default extension of .fir.

Filter designs are always displayed graphically prior to actual construction. Several measurement functions are available, which apply to both classical and arbitrary designs.

Example

Below is an example of an entry in the FIR Lowpass Filter Design block:

Input Design File	firlp
Design Method	KW
Sampling Frequency	48000
Bandwidth	4000
Transition Band	500
Stopband Attenuation	40
Maximum Ripple	1

Output

See Also

FIR Bandpass Filter Design

FIR Bandstop Filter Design

FIR Highpass Filter Design

IIR Bandpass Filter Design

IIR Bandstop Filter Design

IIR Highpass Filter Design

IIR Lowpass Filter Design

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_firlp
Label	FIR Lowpass Filter Design
Data Type	DSP
Block Type	FILT
Attributes	DISP | FIR
Defines	
Symbol Filename	lowpass
Help Filename	refguide
Graphical Control Interface Filename	i_filthb, i_filt
Code Files	c_bfilt, c_afilt
Function Files	f_filter, f_fft

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

IIR Bandpass Filter Design

This block allows construction of IIR Bandpass filters using the Butterworth, Chebyshev I and II, Bessel and Elliptic design methods.

Also, arbitrary designs, such as multiband and irregular or log frequency passbands, can be specified and realized using the Steiglitz method. Selectable filter name, sampling frequency, center frequency, bandwidth, transition bandwidths, stopband attenuation, and passband ripple or corner attenuation. In the case of Steiglitz design, the filter order and required minimum convergence error are also selectable. All designs are first displayed and can be measured in a graphical filter display prior to design; arbitrary designs can be further edited directly using the mouse. Resulting frequency responses can be compared in the graphical display to original specifications and "tolerance regions". Current maximum filter orders are 20 for Bessel designs, 40 for Butterworth, Chebyshev I and II, elliptic and Steiglitz designs.

The software creates a time-domain impulse response waveform as output. Also, the software creates both an ASCII "filter coefficient file", which contains a numerator (pole) section and a denominator (zero) section, and an ASCII "design goal file", which contains the filter specifications and the

arbitrary frequency response, if any.

Inputs	Entry	Description
Input Design File	[path]**filename**[ext]	filter design goal file
Parameters		
Design Method	**B/C1/C2/Bes/E**	specifies the design method to be used. B = Butterworth, C1 = Chebyshev I, C2 = Chebyshev II, Bes = Bessel, E = Elliptic
Sampling Frequency	**n** Hz; >0	specifies sampling frequency in Hz
Center Frequency	**n** Hz, >0	Specifies the center frequency
Bandwidth	**n** Hz, >0	specifies the bandwidth required in Hz
Left Transition Band	**n** Hz, >0	specifies the left transition band in Hz
Right Transition Band	**n** Hz, >0	specifies the right transition band in Hz
Stopband Attenuation	**n** dB, 1..100	specifies stopband attenuation in dB
Maximum Ripple	**n** dB	specifies maximum passband ripple in dB
Outputs		
Output Filter Coefficient File	[path]**filename**[ext]	output filter coefficient file

Comments

The minimum of the transition bandwidth fields is used to determine the filter order or length. Since classical designs always produce symmetrical filters, the minimum transition bandwidth will control both transition bandwidths. Arbitrary designs can be used to achieve asymmetrical transition bandwidths.

If no extension is specified for the Input filter design filename, a .ifd extension is assumed. The output filter coefficient filename has a default extension of .iir.

Example

Below is an example of an entry in the IIR Bandpass Filter Design block:

Input Design File	elliptic
Design Method	E
Sampling Frequency	48000
Center Frequency	12000
Bandwidth	4000
Left Transition band	500
Right Transition Band	500
Stopband Attenuation	40
Maximum Ripple	1
Output	

See Also

FIR Bandpass Filter Design

FIR Bandstop Filter Design

FIR Highpass Filter Design

FIR Lowpass Filter Design

IIR Bandstop Filter Design

IIR Highpass Filter Design

IIR Lowpass Filter Design

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_iirbp
Label	IIR Bandpass Filter Design
Data Type	DSP
Block Type	FILT
Attributes	DISP | IIR
Defines	
Symbol Filename	bandpass
Help Filename	refguide

Graphical Control Interface Filename	i_filtfb, i_filt
Code Files	c_bfilt, c_afilt
Function Files	f_filter, f_ftt

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters.

IIR Bandstop Filter Design

This block allows construction of IIR Bandstop filters using the Butterworth, Chebyshev I and II, Bessel and Elliptic design methods. See IIR Bandpass Filter Design for more information about filter design capabilities of the software.

Inputs	**Entry**	**Description**
Input Design File	[path]**filename**[ext]	filter design goal file
Parameters		
Design Method	**B/C1/C2/Bes/E**	specifies the design method to be used. B = Butterworth, C1 = Chebyshev I, C2 = Chebyshev II, Bes = Bessel, E = Elliptic
Sampling Frequency	**n** Hz; >0	specifies sampling frequency in Hz
Center Frequency	**n** Hz, >0	Specifies the center frequency
Bandwidth	**n** Hz, >0	specifies the bandwidth required in Hz
Left Transition Band	**n** Hz, >0	specifies the left transition band in Hz
Right Transition Band	**n** Hz, >0	specifies the right transition band in Hz
Stopband Attenuation	**n** dB, 1..100	specifies stopband attenuation in dB
Maximum Ripple	**n** dB	specifies maximum passband ripple in dB
Outputs		
Output Filter Coefficient File	[path]**filename**[ext]	output filter coefficient file

Comments

The minimum of the transition bandwidth fields is used to determine the filter order or length. Since classical designs always produce symmetrical filters, the minimum transition bandwidth will control both transition bandwidths. Arbitrary designs can be used to achieve asymmetrical transition bandwidths.

If no extension is specified for the Input filter design filename, a .ifd extension is assumed. The output filter coefficient filename has a default extension of .iir.

Example

Below is an example of an entry in the IIR Bandstop Filter Design block:

Input Design File	iirbstop
Design Method	C2
Sampling Frequency	48000
Center Frequency	12000
Bandwidth	4000
Left Transition band	500
Right Transition Band	500
Stopband Attenuation	40
Maximum Ripple	1
Output	

See Also

FIR Bandpass Filter Design

FIR Bandstop Filter Design

FIR Highpass Filter Design

FIR Lowpass Filter Design

IIR Bandpass Filter Design

IIR Highpass Filter Design

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_iirbs
Label	IIR Bandstop Filter Design
Data Type	DSP
Block Type	FILT
Attributes	DISP | IIR
Symbol Filename	bandstop
Help Filename	refguide
Graphical Control Interface Filename	i_filtfb, i_filt
Code Files	c_bfilt, c_afilt
Function Files	f_filter, f_fft

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

IIR Highpass Filter Design

This block allows construction of IIR Highpass filters using the Butterworth, Chebyshev I and II, Bessel and Elliptic design methods. See IIR Bandpass Filter Design for more information about filter design capabilities of the software.

Inputs	**Entry**	**Description**
Input Design File	[path]**filename**[ext]	filter design goal file
Parameters		
Design Method	**B/C1/C2/Bes/E**	specifies the design method to be used. B = Butterworth, C1 = Chebyshev I, C2 =

		Chebyshev II, Bes = Bessel, E = Elliptic
Sampling Frequency	**n** Hz; >0	specifies sampling frequency in Hz
Bandwidth	**n** Hz, >0	specifies the bandwidth required in Hz
Transition Band	**n** Hz, >0	specifies the left transition band in Hz
Stopband Attenuation	**n** dB, 1..100	specifies stopband attenuation in dB
Maximum Ripple	**n** dB	specifies maximum passband ripple in dB

Outputs

Output Filter Coefficient File	[path]**filename**[ext]	output filter coefficient file

Comments

The minimum of the transition bandwidth fields is used to determine the filter order or length. Since classical designs always produce symmetrical filters, the minimum transition bandwidth will control both transition bandwidths. Arbitrary designs can be used to achieve asymmetrical transition bandwidths.

If no extension is specified for the Input filter design filename, a .ifd extension is assumed. The output filter coefficient filename has a default extension of .iir.

Example

Below is an example of an entry in the IIR Highpass Filter Design block:

Input Design File	iirhp
Design Method	C1
Sampling Frequency	48000
Bandwidth	4000
Transition band	500
Stopband Attenuation	40
Maximum Ripple	1

Output

See Also

FIR Bandpass Filter Design

FIR Bandstop Filter Design

FIR Highpass Filter Design

FIR Lowpass Filter Design

IIR Bandpass Filter Design

IIR Bandstop Filter Design

IIR Lowpass Filter Design

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_iirhp
Label	IIR Highpass Filter Design
Data Type	DSP
Block Type	FILT
Attributes	DISP | IIR
Defines	
Symbol Filename	highpass
Help Filename	refguide
Graphical Control Interface Filename	i_filthb, i_filt
Code Files	c_bfilt, c_afilt
Function Files	f_filter, f_fft

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

IIR Lowpass Filter Design

This block allows construction of IIR Lowpass filters using the Butterworth, Chebyshev I and II, Bessel and Elliptic design methods. See IIR Bandpass Filter Design for more information about filter design capabilities of the software.

Inputs	Entry	Description
Input Design File	[path]**filename**[ext]	filter design goal file
Parameters		
Design Method	**B/C1/C2/Bes/E**	specifies the design method to be used. B = Butterworth, C1 = Chebyshev I, C2 = Chebyshev II, Bes = Bessel, E = Elliptic
Sampling Frequency	**n** Hz; >0	specifies sampling frequency in Hz
Bandwidth	**n** Hz, >0	specifies the bandwidth required in Hz
Transition Band	**n** Hz, >0	specifies the left transition band in Hz
Stopband Attenuation	**n** dB, 1..100	specifies stopband attenuation in dB
Maximum Ripple	**n** dB	specifies maximum passband ripple in dB
Outputs		
Output Filter Coefficient File	[path]**filename**[ext]	output filter coefficient file

Comments

The minimum of the transition bandwidth fields is used to determine the filter order or length. Since classical designs always produce symmetrical filters, the minimum transition bandwidth will control both transition bandwidths. Arbitrary designs can be used to achieve asymmetrical transition bandwidths.

If no extension is specified for the Input filter design filename, a .ifd extension is assumed. The output filter coefficient filename has a default extension of .iir.

Example

Below is an example of an entry in the IIR Lowpass Filter Design block:

Input Design File	iirhp
Design Method	C1
Sampling Frequency	48000
Bandwidth	4000
Transition band	500
Stopband Attenuation	40
Maximum Ripple	1
Output	

See Also

FIR Bandpass Filter Design

FIR Bandstop Filter Design

FIR Highpass Filter Design

FIR Lowpass Filter Design

IIR Bandpass Filter Design

IIR Bandstop Filter Design

IIR Lowpass Filter Design

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_iirlp
Label	IIR Lowpass Filter Design
Data Type	DSP
Block Type	FILT

Attributes	DISP | IIR
Defines	
Symbol Filename	lowpass
Help Filename	refguide
Graphical Control Interface Filename	i_filthb, i_filt
Code Files	c_bfilt, c_afilt
Function Files	f_filter, f_fft

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

1.5 Function Blocks

1.5.1 Signal Processing Blocks

Signal Processing Functions available in the DSPower system library are listed below. For a complete listing of blocks in alphabetical order, see Block Alphabetic Listing and Cross-Reference, above.

Signal Processing Blocks

Circular Correlation

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1

Input 2	[path]**filename**[ext]	waveform to display on trace 2

Outputs

Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2

Parameters

Comments

Comments go here.

Example

Below is an example of an entry in the Circular Correlation block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_crcorr
Label	Circular Correlation
Data Types	DSP
Block Types	FUNC | SP
Attributes	0

Defines	
Symbol Filename	conv
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_bfunc, c_afunc
Function Files	f_conv

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Convolution

Convolution has selectable input waveform, filter waveform, and output waveform. Filter waveforms can be output of standard block functions such as FIR and IIR Filter Construction and Spectrum Analyzer transfer function storage, or any arbitrary time domain waveform. Performed by the host or by the currently selected DSP hardware when enabled as a "DSP Accelerator."

Inputs	**Entry**	**Description**
Input File	[path]**filename**[ext]	input file
Filter File	[path]**filename**[ext]	filter file
Outputs		
Output File	[path]**filename**[ext]	output file

Comments

This function uses an external program for filter sizes greater than 16384 point. Again, this could vary depending on the DSP Engine being used. Input waveform size is not limited.

If no extension is specified for the files, a .tim extension is assumed.

Example

Below is an example of an entry in the Convolution block:

Input File	speech
Filter File	kaiser
Output File	sp_out

See Also

Filter Blocks

Deconvolution

Correlation

Circular Correlation

Wavelet Transform

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_conv
Label	Convolution
Data Type	DSP
Block Type	FUNC | SP | FILT
Attributes	GENEXT
Defines	
Symbol Filename	conv
Help Filename	refguide
Graphical Control Interface Filename	
Code Files	c_bfunc, c_afunc
Function Files	f_conv

For more information on Registration Information, see Signalogic DSP Software Users

Guide section , Registration Parameters

Correlation

[@k Circular Correlation, Cross Correlation, correlation,, circular]

Similar to Convolution, Correlation and Circular Correlation also have selectable input waveforms, filter waveforms, and output waveforms. Filter waveforms can be output of standard block functions such as FIR and IIR Filter Construction and Spectrum Analyzer transfer function storage, or any arbitrary time domain waveform. Performed by the host or by the currently selected DSP hardware when enabled as a "DSP Accelerator."

Inputs	**Entry**	**Description**
Input File	[path]**filename**[ext]	input file
Filter File	[path]**filename**[ext]	filter file
Outputs		
Output File	[path]**filename**[ext]	output file

Comments

These functions use an external program for filter sizes greater than 16384 point. Again, this could vary depending on the DSP Engine being used. Input waveform size is not limited.

If no extension is specified for the files, a .tim extension is assumed.

Example

Below is an example of an entry in the Correlation/Circular Correlation block:

Input File	speech
Filter File	kaiser
Output File	sp_out

See Also

Filter Blocks

Deconvolution

Convolution

Circular Correlation

Wavelet Transform

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_corr
Label	Correlation
Data Type	DSP
Block Type	FUNC | SP
Attributes	GENEXT
Defines	
Symbol Filename	conv
Help Filename	refguide
Graphical Control Interface Filename	
Code Files	c_bfunc, c_afunc
Function Files	f_conv

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Deconvolution

Description Text.

Inputs	Entry	Description
Input 1	[path]filename[ext]	waveform to display on trace 1
Input 2	[path]filename[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]filename[ext]	waveform output from trace 1
Output 2	[path]filename[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Deconvolution block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_xfer
Label	Deconvolution
Data Types	DSP
Block Types	FUNC | SP
Attributes	0
Defines	
Symbol Filename	dconv
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_bfunc, c_afunc
Function Files	f_xfer

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Delay

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Delay block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_delay
Label	Delay
Data Types	DSP
Block Types	FUNC | SP
Attributes	0
Defines	
Symbol Filename	delay
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_bfunc, c_afunc
Function Files	f_delay

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Difference Equations

See Difference Equations in section 1.5.2, Math Blocks.

FFT Generation

FFT allows selectable input waveform and choosing the Ex option allows external parameters like framesize, frame overlap, Window type etc. to be specified.

Depending on the DSP Engine used, an external program is used for FFT sizes > 4096 points.

Framesize entry can be used to cause segmentation of the input waveform. When framesize is less than FFT size, remaining FFT input points are zero-filled. A ‘0’ entry causes framesize to be determined automatically from the input waveform file header.

FFT size should be entered as a value equal to 2 raised to an integer power, ranging from 4 to 262,144. Non-power-of-two values are quantized to the nearest allowable value. A ‘0’ entry causes FFT size to be determined automatically by the framesize entry (first power of two value greater or equal to the framesize). Larger FFT sizes may not run, or not run efficiently, depending on the amount of extended memory in the host machine.

Currently, this block does not allow overlap specification or window selection for FFT sizes > 4096. If using the EX option, this function

does not make use of DSP accelerator hardware.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the FFT Generation block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_fft
Label	FFT Generation
Data Types	DSP
Block Types	FUNC | SP
Attributes	GENEXT
Defines	
Symbol Filename	fft
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_bfunc, c_afunc
Function Files	f_fft

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Filter Asm Code Generation

Description Text

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Filter Asm Code Generation block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_asmgen
Label	Filter Asm Code Generation
Data Types	DSP
Block Types	FILT
Attributes	EXEC | DISP | FIR | IIR
Defines	
Symbol Filename	asmgen
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_bfunc, c_afunc
Function Files	f_asmgen

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Frequency Zoom

This block takes the input waveform, multiplies by complex exponential to shift zoom area to baseband (origin), then performs sampling rate conversion to remove unwanted bandwidth and obtain correct new sampling rate, and finally performs manipulation in complex frequency domain and iFFT to create real-valued time domain waveform as output.

Inputs	**Entry**	**Description**
Input File	[path]**filename**[ext]	input file
Parameters		
Normalized Start Frequency	**n**	specifies Normalized Start Frequency
Normalized Stop Frequency	**n**	specifies Normalized Stop Frequency
Outputs		
Output File	[path]**filename**[ext]	output file

Comments

The start and stop frequencies should be entered so that, when multiplied by half the sampling rate, more or less an integer results.

If no extension is specified for the files, a .tim extension is assumed.

Example

Below is an example of an entry in the Frequency Zoom block:

Input File	signal
Normalized Start Frequency	0.25
Normalized Stop Frequency	0.75
Output File	s_out

See Also

Sampling Rate Conversion

Fs Decimation (Divide)

Fs Interpolation (Multiply)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_fzoom
Label	Frequency Zoom
Data Type	DSP
Block Type	FUNC | SP
Attributes	0
Defines	
Symbol Filename	interp
Help Filename	refguide
Graphical Control Interface Filename	
Code Files	c_bfunc, c_afunc
Function Files	f_fzoom

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Fs Decimation (Divide)

This block can be used if the sampling rate of the input needs to be reduced. The divide factor is one of the parameters for this block.

Inputs	**Entry**	**Description**
Input File	[path]**filename**[ext]	input file
Parameters		
Decimation Integer	**n**	specifies the divide factor
Outputs		
Output File	[path]**filename**[ext]	output file

Comments

If no extension is specified for the files, a .tim extension is assumed.

Example

Below is an example of an entry in the Decimation block:

Input File	speech
Normalized Start Frequency	7
Normalized Stop Frequency	2
Output File	sp_out

See Also

Frequency Zoom

Fs Decimation (Divide)

Fs Interpolation (Multiply)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_deci
Label	Fs Decimation
Data Type	DSP
Block Type	FUNC | SP
Attributes	0
Defines	
Symbol Filename	deci
Help Filename	refguide
Graphical Control Interface Filename	
Code Files	c_bfunc, c_afunc
Function Files	f_fsconv

For more information on Registration Information, see Signalogic DSP Software Users

Guide section , Registration Parameters

Fs Interpolation (Mult)

[@k sampling frequency interpolation, sampling interpolation, interpolation,, sampling]

This block can be used if the sampling rate of the input needs to be increased. The multiply factor is one of the parameters for this block.

Inputs	Entry	Description
Input File	[path]**filename**[ext]	input file
Parameters		
Interpolation Integer	**n**	specifies the multiply factor
Outputs		
Output File	[path]**filename**[ext]	output file

Comments

If no extension is specified for the files, a .tim extension is assumed.

Example

Below is an example of an entry in the Interpolation block:

Input File	speech
Interpolation factor	7
Output File	sp_out

See Also

Frequency Zoom

Fs Decimation (Divide)

Fs Interpolation (Multiply)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

Registration Information

Template Filename	t_interp
Label	Fs Interpolation (Mult)
Data Type	DSP
Block Type	FUNC | SP
Attributes	0
Defines	
Symbol Filename	interp
Help Filename	refguide
Graphical Control Interface Filename	
Code Files	c_bfunc, c_afunc
Function Files	f_fsconv

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

iFFT

Using real and imaginary data files as input, frequency waveform files undergo a fast inverse Fourier transform to create an output time domain waveform.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the iFFT block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_ifft
Label	iFFT
Data Types	DSP
Block Types	FUNC | SP
Attributes	0
Defines	
Symbol Filename	ifft
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_bfunc, c_afunc
Function Files	f_ifft

For more information on Registration Information, see Signalogic DSP Software Users

Guide section , Registration Parameters

LPC Autocorrelation

Description Text

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the LPC Autocorrelation block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_lpc
Label	LPC Autocorrelation
Data Types	DSP
Block Types	FUNC | SP
Attributes	GENEXT
Defines	
Symbol Filename	lpc
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_bfunc, c_afunc
Function Files	f_lpc

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Power Spectra Estimation

[@k power spectra estimation, estimation,, power spectra]

Power spectra waveform files are calculated essentially from one or more input magnitude frames averaged together or filtered producing one or more output frames. Output is always "root-mean-square"; To see power spectra generation results, use any of the Power spectra display or use the Spectrum Analyzer and enable the power-spec field;

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2

Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2

Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Power Spectra Estimation block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_pwrgen
Label	Power Spectra Estimation
Data Types	DSP
Block Types	FUNC | SP
Attributes	0

Defines	
Symbol Filename	pwrspec
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_bfunc, c_afunc
Function Files	f_pwrgen

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Running Sum (Boxcar Filter)

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Running Sum block:

Input 1 speech
Input 2
Output 1

Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_runsum
Label	Running Sum (Boxcar Filter)
Data Types	DSP
Block Types	FUNC | SP | FILT
Attributes	0
Defines	
Symbol Filename	runsum
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_bfunc, c_afunc
Function Files	f_diffeq

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Recursive Filtering

Applies IIR filters in input data. Selectable input waveform, filter coefficient file, and output waveform. The input "IIR Filter Coefficient" files are in ASCII format and can be standard output

from the Hypersignal Filter functions, or user-defined. The coefficient file format accepts both factored and direct-form numerator (poles) and denominator (zeros), and can be applied in "biquad cascade" (factored) form or direct form.

Inputs	Entry	Description
Input File	[path]**filename**[ext]	input file
Parameters		
Interpolation Integer	**n**	specifies the multiply factor
Decimation Integer	**n**	specifies the divide factor
Outputs		
Output File	[path]**filename**[ext]	output file

Comments

If no extension is specified for the files, a .tim extension is assumed.

Example

Below is an example of an entry in the Recurvise Filtering block:

Input File	speech
Normalized Start Frequency	7
Normalized Stop Frequency	2
Output File	sp_out

See Also

Frequency Zoom

Fs Decimation (Divide)

Fs Interpolation (Multiply)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_recur
Label	Recursive Filtering
Data Type	DSP
Block Type	FUNC | SP | FILT
Attributes	GENEXT | IIR
Defines	
Symbol Filename	recur
Help Filename	refguide
Graphical Control Interface Filename	
Code Files	c_bfunc, c_afunc
Function Files	f_recur

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Sampling Rate Conversion

[@k sampling rate conversion, conversion,, sampling]

This block can be used either to reduce or increase the sampling rate of a given waveform. It takes in both multiply and divide integers as parameters. It initially multiplies the waveform with the multiple integer and then divides with the divide factor.

Inputs	Entry	Description
Input File	[path]**filename**[ext]	input file
Parameters		
Interpolation Integer	**n**	specifies the multiply factor
Decimation Integer	**n**	specifies the divide factor
Outputs		
Output File	[path]**filename**[ext]	output file

Comments

If no extension is specified for the files, a .tim extension is assumed.

Example

Below is an example of an entry in the Sampling Rate Conversion block:

Input File	speech
Normalized Start Frequency	7
Normalized Stop Frequency	2
Output File	sp_out

See Also

Frequency Zoom

Fs Decimation (Divide)

Fs Interpolation (Multiply)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

section 5.4 Interactive Operation

Registration Information

Template Filename	t_
Label	
Data Type	DSP
Block Type	
Attributes	
Defines	
Symbol Filename	
Help Filename	refguide
Graphical Control Interface Filename	
Code Files	
Function Files	f_

For more information on Registration Information, see Signalogic DSP Software Users

Guide section , Registration Parameters

Wavelet Transform, Continuous

[@k continuous wavelet transform]

The Wavelet block accepts the following input parameters: Input filename, Wavelet (file or AUTO mode), Output filename and number of constant scale values.

Choosing the wavelet:

wlt.mac accepts the following as input parameters to the wavelet field:

- A complex wavelet file;
- A real wavelet file (complex wavelet is created automatically)
- "autosegment" option, if wavelet is to be derived from signal itself, can be entered as 'offset,length', in samples. For example, an entry of '75,32' in this field would cause the program to use a 32 point wavelet, derived from samples 75 through 106 of the input waveform.

Notes

1) When using the autosegment option, it is recommended to choose a periodic portion of the input signal.

Inputs	**Entry**	**Description**
Input File	[path]**filename**[ext]	input file
Wavelet	[path]**filename**[ext], **n1, n2**	wavelet filename, Use n1= starting point, n2 = ending point to create wavelet from input.
Parameters		
Number of Scale Values	**n** points	Number of constant scale values
Outputs		
Output File	[path]**filename**[ext]	output file

Comments

The example wavelet provided with this block is a Gaussian wavelet derived from samples 75 through 106 of the 3-component signal also provided with this macro; Note that the last component has a center frequency of pi/3 radians (6 samples per period)

This block produces frequency domain output compatible with the displays of the DSP Engine being used. Currently, this function does not allow the input waveform to be segmented. So as a result, this function does not allow large input waveforms.

If no extension is specified for the files, a .tim extension is assumed.

Example

Below is an example of an entry in the Wavelet Transform block:

Input File	sig
Wavelet	wave
Number of scale values	128
Output File	sig_out

See Also

Filter Blocks

Convolution

Deconvolution

Correlation

Circular Correlation

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.1.4 Display Settings

Registration Information

Template Filename	t_wlt
Label	Wavelet Transform, Continuous
Data Type	DSP
Block Type	FUNC | SP
Attributes	0
Defines	
Symbol Filename	wlt
Help Filename	refguide
Graphical Control Interface Filename	
Code Files	c_bfunc, c_afunc
Function Files	f_wlt

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

1.5.2 Math Blocks

Math functions available in the DSPower system library are listed below. For a complete listing of blocks in alphabetical order, see Block Alphabetic Listing and Cross-Reference, above.

Math Blocks

Add2

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace

		2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Add2 block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_add2
Label	Add2
Data Types	DSP
Block Types	FUNC | MATH
Attributes	0
Defines	

Symbol Filename	add2
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_bfunc, c_afunc
Function Files	f_diffeq

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Difference Equations

Generation of new waveforms and manipulation of existing waveforms using various operators and functions can be specified in difference equation form. Input sources include existing waveforms, unit-step and delta functions, and previous output (recursive) data. Operators include signal arithmetic, multiplication, division (and integer division) modulus, differentiation, integration, min and max, and a frequency domain "frame operator." Functions include several trigonometric and transcendental functions, random noise, and various linear and non-linear math functions.

Equations are based on a discrete time index "n", which begins at zero and increments a specified number of times, according to user entry. The difference equation evaluates at each discrete time index. Note that it is also possible to write expressions for n, both

when used as an input index, on the left side of the equals sign.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Difference Equations block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_diffeq
Label	Difference Equations
Data Types	DSP
Block Types	FUNC | SP | MATH | GEN
Attributes	GENEXT
Defines	
Symbol Filename	diffeq
Help Filename	refguide
Graphical Control Interface Filenames	i_diffeq
Code Files	c_bfunc, c_afunc
Function Files	f_diffeq

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Mix (Heterodyne)

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Mix block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_mix
Label	Mix (Heterodyne)
Data Types	DSP
Block Types	FUNC | MATH
Attributes	0
Defines	
Symbol Filename	mult2
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_bfunc, c_afunc
Function Files	f_diffeq

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Mult2

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Mult2 block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_mult2

Label	Mult2
Data Types	DSP
Block Types	FUNC | MATH
Attributes	0
Defines	
Symbol Filename	mult2
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_bfunc, c_afunc
Function Files	f_diffeq

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Scale Factor

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Scale Factor block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_scale
Label	Scale Factor
Data Types	DSP
Block Types	FUNC | MATH
Attributes	0
Defines	
Symbol Filename	scale
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_bfunc, c_afunc
Function Files	f_diffeq

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Sub2

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Sub2 block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_sub2
Label	Sub2

Data Types	DSP
Block Types	FUNC | MATH
Attributes	0
Defines	
Symbol Filename	sub2
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_bfunc, c_afunc
Function Files	f_diffeq

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

1.5.3 Signal Generation and Source Blocks

Signal Generation functions available in the DSPower system library are listed below. For a complete listing of blocks in alphabetical order, see Block Alphabetic Listing and Cross-Reference, above.

Signal Generation/Source Blocks

Cosine Generator

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace

		2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Cosine Generation block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_cosgen
Label	Cosine Generator
Data Types	DSP
Block Types	FUNC | GEN
Attributes	0
Defines	

Symbol Filename	cosgen
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_bfunc, c_afunc
Function Files	f_diffeq

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Difference Equations

See Difference Equations in section 1.5.2, Math Blocks.

Sawtooth Generator

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Sawtooth Generator block:

Input 1 speech

Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_trigen
Label	Sawtooth Generator
Data Types	DSP
Block Types	FUNC | GEN
Attributes	0
Defines	
Symbol Filename	trigen
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_bfunc, c_afunc
Function Files	f_diffeq

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Sine Generator

Description Text.

Inputs	Entry	Description

Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2

Outputs

Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2

Parameters

Comments

Comments go here.

Example

Below is an example of an entry in the Sine Generator block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_singen
Label	Sine Generator
Data Types	DSP

Block Types	FUNC | GEN
Attributes	0
Defines	
Symbol Filename	singen
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_bfunc, c_afunc
Function Files	f_diffeq

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Squarewave Generator

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Squarewave Generator block:

Input 1 speech

Input 2

Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_sqrgen
Label	Squarewave Generator
Data Types	DSP
Block Types	FUNC | GEN
Attributes	0
Defines	
Symbol Filename	sqrgen
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_bfunc, c_afunc
Function Files	f_diffeq

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Waveform File Source

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace

		1
Input 2	[path]**filename**[ext]	waveform to display on trace 2

Outputs

Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2

Parameters

Comments

Comments go here.

Example

Below is an example of an entry in the Waveform File Source block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_wvfsrc
Label	Waveform File Source
Data Types	DSP
Block Types	FUN | GEN

Attributes	0
Defines	
Symbol Filename	wvfsrc
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	
Function Files	

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

1.6 Instrument Blocks

Instrumentation functions available in the DSPower system library are listed below. For a complete listing of blocks in alphabetical order, see Block Alphabetic Listing and Cross-Reference, above.

Instrument Blocks

2-Chan Transfer Analyzer

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace

Comments

Comments go here.

Example

Below is an example of an entry in the 2-Chan Transfer Analyzer block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_xfrana
Label	2-Chan Transfer Analyzer
Data Types	DSP
Block Types	INST
Attributes	DISP
Defines	XFRANA
Symbol Filename	spcana
Help Filename	refguide
Graphical Control Interface Filenames	i_spcana

Code Files	c_ainst
Function Files	f_spcana

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

A/D Input

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the A/D Input block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_ad
Label	A/D Input
Data Types	DSP
Block Types	INST
Attributes	0
Defines	
Symbol Filename	ADin
Help Filename	refguide
Graphical Control Interface Filenames	i_ad
Code Files	c_ainst
Function Files	f_anlcnv

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Continuous Signal Gen

[@k continuous signal generation, generation,, continuous, signal synthesis, synthesis,, signal]

Signal generation looping in the Analog Conversion menu item has been enhanced to allow easier construction of a continuous function generator. By creating short signal/function waveforms with the Difference Equations function, and then using these waveforms as input to the Analog

Conversion function, a continuous signal generator can be achieved when the ACQ/GEN field is set to 'GL'. Some guidelines to follow when specifying a Difference Equation:

- waveform lengths (as specified in the Difference Equations # OF SAMPLES field) should be an integer multiple of the function period; if you are in doubt about the function period, use the Waveform Display/Edit option to measure it exactly
- the overall waveform length should be anywhere from a few hundred samples to less than 6000 samples; only one or just a few periods of the function waveform is not recommended, because of the overhead required by the DSP software when "wrapping around" to repeat the function. And, definitely, long waveforms are not recommended because they become split into buffer segments due to DSP board memory limitations, defeating the continuous mechanism altogether
- the length of the waveform, in samples (and multiplied by the number of channels), should be divisible by two
- watch out for sampling frequency when designing the difference equation; most hardware only allows certain sampling frequencies, and if you base your calculations on a sampling frequency the hardware cannot achieve, then there is likely to be a discontinuity when the output waveform repeats

Inputs	**Entry**	**Description**
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Continuous Signal Gen block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_siggen
Label	Continuous Signal Gen
Data Types	DSP
Block Types	INST
Attributes	INST
Defines	
Symbol Filename	siggen
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_ainst
Function Files	f_diffeq, f_anlcnv

For more information on Registration Information, see Signalogic DSP Software Users

Guide section , Registration Parameters

D/A Output

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the D/A Output block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_da
Label	D/A Output
Data Types	DSP
Block Types	INST
Attributes	0
Defines	
Symbol Filename	DAout
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_ainst
Function Files	f_anlcnv

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Digital Oscilloscope

See Digital Oscilloscope in section 1.3, Display Blocks.

Digital Tape Recorder

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Digital Tape Recorder block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_dtape
Label	Digital Tape Recorder
Data Types	DSP
Block Types	INST
Attributes	INST | EXEC
Defines	
Symbol Filename	dtape
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_ainst
Function Files	f_exec

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Gen Stimulus and Response

[@k general stimulus and reponse, abitrary stimulus and response]

A Single-channel or two-channel stimulus and response measurement function is available that allows simultaneous output of stimuli waveform and continuous recording of analog input disk. Controls include sampling rate, hardware gain (if programmable), triggering (channel and level), software scaling and offsets, amount of data, and positive or negative stimulus delay.

Inputs	**Entry**	**Description**
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Gen Stimulus and Response block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_stim
Label	Gen Stimulus and Response
Data Types	DSP
Block Types	INST
Attributes	DISP | EDIT
Defines	STIMRESP
Symbol Filename	stim
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_ainst
Function Files	f_anlcnv

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

MLS Stimulus and Response

This function is still considered to be in a beta state, with improvements continuing to be added.

This function prompts for stimulus length, number of averages, output waveform filename, sampling frequency, a channel list, and stimulus amplitude and offset. It uses the currently installed hardware specified in the System Configuration menu ANALOG CONVERSION field, performs the stimulus & response operation, synchronously averages the results in the time domain, performs a cross-correlation operation to extract the impulse response, and then an FFT to calculate the frequency response. The macro function concludes by showing the frequency response in the Magnitude Display, with vertical scale set to +/- 50 dB.

When using this function, the stimulus length should be entered in seconds. The actual stimulus length in samples is chosen by taking the entered value, multiplying it by the entered sampling rate, and quantizing the result up to the nearest allowable MLS length. Quantization is due to the nature of the MLS process, which requires stimulus lengths to be a power of 2 minus 1; for example, 4095, 8191, 16383, and 65535. BE AWARE that some DSP hardware can

achieve only certain sampling rates, so it is a good idea to know these ahead of time if you are trying to calculate stimulus lengths exactly. See the Hardware Reference Guide for information about memory and sampling rate limitations for individual hardware types.

Currently, mls.mac does not have an option to equalize out the response due solely to the measurement board being used. This option, and other “calibration mode” features, are expected to be added later.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the MLS Stimulus and Response block:

Input 1 speech
Input 2
Output 1

Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_mls
Label	MLS Stimulus and Response
Data Types	DSP
Block Types	INST
Attributes	DISP | MAGDSP | FRQAXIS | FRQZOOM | FRQPAN
Defines	STIMRESP
Symbol Filename	stim
Help Filename	refguide
Graphical Control Interface Filenames	i_file, i_wvfrm, i_xaxis, i_yaxis, i_dspopt, i_dsphlp
Code Files	c_ainst
Function Files	f_diffeq, f_mls, f_fft, f_magdsp

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Octave Band Analyzer

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1

Input 2	[path]**filename**[ext]	waveform to display on trace 2

Outputs

Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2

Parameters

Comments

Comments go here.

Example

Below is an example of an entry in the Octave Band Analyzer block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_octana
Label	Octave Band Analyzer
Data Types	DSP
Block Types	INST
Attributes	DISP

Defines	OCTANA
Symbol Filename	octana
Help Filename	refguide
Graphical Control Interface Files	i_spcana
Code Files	c_ainst
Function Files	f_spcana

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Spectrum Analyzer

The Spectrum Analyzer can display single-, dual-, or overlaid-trace input from previously stored waveform files or any of the currently active analog channel inputs. Various combinations are possible, including viewing the same analog channel in different domains, formats, or with different filtering options, and overlaying analog inputs with previously stored waveform file input. Selectable display data formats include time, linear/log magnitude, phase, transfer function, and impulse response (iFFT of transfer function). Averaging can be enabled. Operation from the DSP chip point of view is always in real-time up to the maximum sampling rate, but the rate of update of display frames to the video system of the host computer depends on the speed of the host I/O bus and video system. The maximum sampling rate possible depends on the DSP/Acquisition board and DSP chip in use.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Spectrum Analyzer block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_spcana
Label	Spectrum Analyzer

Data Types	DSP
Block Types	INST | DISP
Attributes	DISP
Defines	SPCANA
Symbol Filename	spcana
Help Filename	refguide
Graphical Control Interface Filenames	i_spcana
Code Files	c_ainst
Function Files	f_spcana

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Strip Chart Recorder

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Strip Chart Recorder block:

Input 1 speech

Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_strip
Label	Strip Chart Recorder
Data Types	DSP
Block Types	INST
Attributes	EXEC | INST | DISP
Defines	
Symbol Filename	strip
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_ainst
Function Files	f_exec

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

1.7 Monitor / Debug Blocks

Some general purpose Utility functions available in the DSPower

system library are listed below. For a complete listing of blocks in alphabetical order, see Block Alphabetic Listing and Cross-Reference, above.

Monitor/Debug Blocks

.MAT to .TIM Convert

File Export

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the File Export block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_fileex
Label	File Export
Data Types	DSP
Block Types	UTIL
Attributes	0
Defines	
Symbol Filename	fileex
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_butil, c_autil
Function Files	f_fileex

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

File Import

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace

		1
Output 2	[path]**filename**[ext]	waveform output from trace 2

Parameters

Comments

Comments go here.

Example

Below is an example of an entry in the File Import block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_fileim
Label	File Import
Data Types	DSP
Block Types	UTIL
Attributes	0
Defines	
Symbol Filename	fileim
Help Filename	refguide

Graphical Control Interface Filenames	
Code Files	c_butil, c_autil
Funtion Files	f_fileim

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

Waveform File Conversion

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Waveform File Conversion block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log

Parameter 2 0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_filcnv
Label	Waveform File Conversion
Data Types	DSP
Block Types	UTIL
Attributes	0
Defines	
Symbol Filename	filcnv
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_butil, c_autil
Function Files	f_filcnv

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

1.8 Utility Blocks

Some general purpose Utility functions available in the DSPower system library are listed below. For a complete listing of blocks in alphabetical order, see Block Alphabetic Listing and Cross-Reference, above.

.MAT to .TIM Convert

File Export

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the File Export block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_fileex
Label	File Export
Data Types	DSP
Block Types	UTIL
Attributes	0
Defines	
Symbol Filename	fileex
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_butil, c_autil
Function Files	f_fileex

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

File Import

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the File Import block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

2-D Spectrograph (Contour)

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_fileim
Label	File Import
Data Types	DSP
Block Types	UTIL
Attributes	0
Defines	
Symbol Filename	fileim
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_butil, c_autil
Funtion Files	f_fileim

For more information on Registration Information, see

Signalogic DSP Software Users Guide section , Registration Parameters

Waveform File Conversion

Description Text.

Inputs	Entry	Description
Input 1	[path]**filename**[ext]	waveform to display on trace 1
Input 2	[path]**filename**[ext]	waveform to display on trace 2
Outputs		
Output 1	[path]**filename**[ext]	waveform output from trace 1
Output 2	[path]**filename**[ext]	waveform output from trace 2
Parameters		

Comments

Comments go here.

Example

Below is an example of an entry in the Waveform File Conversion block:

Input 1	speech
Input 2	
Output 1	
Output 2	
Parameter 1	Log
Parameter 2	0.0

Entry Notes

1) Note Text.

See Also

Related Users Guide Topics

section 5.3.1 Creating and Adding Blocks

section 5.4 Interactive Operation

Registration Information

Template Filename	t_filcnv
Label	Waveform File Conversion
Data Types	DSP
Block Types	UTIL
Attributes	0
Defines	
Symbol Filename	filcnv
Help Filename	refguide
Graphical Control Interface Filenames	
Code Files	c_butil, c_autil
Function Files	f_filcnv

For more information on Registration Information, see Signalogic DSP Software Users Guide section , Registration Parameters

2 Function Reference

Overview

The Function Reference section provides technical descriptions of functions available in DSPower series and Hypersignal-Macro series software. Functions are further subdivided into the following categories:

- DSP Engine Functions
- Library Functions
- DSP Source Code Functions

Note that DSP Engine functions are not the same as block functions. Blocks, which begin as template files and become executable DSP Engine programs in the DSPower terminology, are covered separately in chapter 1, Block Reference.

See Also

Function Alphabetic List and Cross Reference

Below is an alphabetic listing of all functions, cross-referenced by page number.

Function Categories

DSP Engine Functions

Library Functions

DSP Source Code Functions

Function Alphabetic List and Cross Reference

The table below provides a cross-reference for all DSPower series library functions to their corresponding Reference Guide descriptions, which contain parameter information and calling instructions, comments, related functions, and examples. Each Reference Guide description also contains a "Related Users Guide Topic" section which lists related areas in the Signalogic DSP Software Users Guide. Users Guide information typically is more general in nature, including more extensive source code examples and explanation of relationship with other functions and software components.

Function Name	Page
DSAcquireWvfrmFile	587
DSAddEngMgrStatusLine	510
DSAssignBoard	587
DSCalcSampFreq	589
DSCancelWaitBuffer	593
DSCancelWaitFlag	593

DSDisableBoard 597
DSDSPToIEEE 601
DSEngineOpen 511
DSEngineClose 514
DSExecEngineFunction 516
DSFreeBoard 600
DSGenerateWvfrmFile 605
DSGetBoardBaseAddr 611
DSGetBoardBusType 606
DSGetBoardClass 609
DSGetBoardInfo 609
DSGetEngineErrorStatus 521
DSGetEnginePath 524
DSGetEngineUsage 525
DSGetEngineState 525
DSGetEngMgrErrorStatus 528
DSGetHVarMem 613
DSGetMem 615
DSGetMemArch 619
DSGetMemSize 626
DSGetHWMgrCurVal 545
DSGetHWMgrEntry 548
DSGetHWMgrEntryIndex 551
DSGetHWMgrErrorStatus 553
DSGetHWMgrNumEntries 555
DSGetHWMgrVar 558
DSGetHWSetupFile 561
DSGetWordLength 631
DSHideEngMgrStatusWindow 532
DSHoldBoard 628
DSHoldProcessor 635
DSInitBoard 637

DSIEEEToDSP 642
DSLoadFileBoard 640
DSLoadFileProcessor 645
DSPutHVarMem 648
DSPutMem 651
DSReadBoardReg 663
DSReadHWSetupFile 561
DSRegisterEngineMsgWnd 533
DSResetBoard 663
DSResetProcessor 664
DSRunBoard 666
DSRunProcessor 669
DSSendEngineCommand 535
DSSetBoardBaseAddr 670
DSSetBoardBusType 670
DSSetEngineTemplatePath 538
DSSetEngineWaveformPath 540
DSSetHWMgrEntry 569
DSSetHWMgrCurVal 567
DSSetHWMgrVar 572
DSSetHWSetupFile 576
DSShowEngMgrStatusWindow 541
DSShowHardwareSelectorDlg 576
DSUnregisterEngineMsgWnd 542
DSWaitForBuffer 675
DSWaitForFlag 677
DSWriteBoardReg 684
DSWriteHWSetupFile 576

2.1 DSP Engine Functions

DSP Engine simulation functions have been further subdivided into the following categories:

- Hypersignal-Macro series software functions
- MATLAB functions

Within each category, functions are listed alphabetically. Note that with the online Help version of documentation, you can use the Browse buttons (">>" and "<<") to move forward and backwards between different function descriptions.

For a complete listing of all DSPower series software functions, see Function Alphabetic List and Cross Reference, above.

NOTE: In function descriptions listed below, knowledge of Hypersignal macro language and MATLAB .m program definition, variable declarations, and syntax is assumed.

DSP Engine Function Categories

2.2 Library Functions

Library functions have been further subdivided into the following categories:

- Engine Manager Functions (enmgr.dll, enmgr32.dll)
- Hardware Manager Functions (hwmgr.dll, hwmgr32.dll)
- Hardware Library Functions (hwlib.dll, hwlib32.dll)
- Signal Processing Library Functions (splib.dll)

Within each category, functions are listed alphabetically. Note that with the online Help version of documentation, you can use the Browse buttons (">>" and "<<") to move forward and backwards between different function descriptions.

For a complete listing of all DSPower series software functions, see Function Alphabetic List and Cross Reference, above.

NOTE: In the function descriptions listed below, knowledge of type definitions declared in the standard Microsoft windows.h include file is assumed. For example, LPCSTR is defined as const char far*, UINT is defined as unsigned int, etc.

Library Function Categories

Related Users Guide Chapters

Using Engine Manager

Using Hardware Manager

Using DSPower-HWLib and DSPower-SPLib Software

2.2.1 Engine Manager Functions

[@k functions, Engine Manager, enmgr.dll, enmgr32.dll]

Below are listed functions in the Engine Manager API. For information on how to include Engine Manager (enmgr.dll or enmgr32.dll) in C/C++, Visual Basic, MATLAB, and LabView projects, see Signalogic DSP Software Users Guide section 7.1, Engine Manager Access.

Engine Manager Functions

DSAddEngMgrStatusLine

DSEngineOpen

DSEngineClose

DSExecEngineFunction

DSGetEngineErrorStatus

DSGetEnginePath

DSGetEngineUsage

DSGetEngineState

DSGetEngMgrErrorStatus

DSHideEngMgrStatusWindow

DSRegisterEngineMsgWnd

DSSendEngineCommand

DSSetEngineTemplatePath

DSSetEngineWaveformPath

DSShowEngMgrStatusWindow

DSUnregisterEngineMsgWnd

DSAddEngMgrStatusLine

HWND DSAddEngMgrStatusLine (pszText)

LPCSTR pszText; // pointer to text string to add to error/status window

The DSAddEngMgrStatusLine function adds a line of text to the Engine Manager error/status listbox. Text is always added to the end of any text currently in the listbox, regardless of whether the error/status listbox is currently enabled or not.

Parameter	Description
pszText	pointer to null-terminated text string to add to Engine Manager error/status listbox

Returns

The return value is the handle of the error/status listbox window, if successful. Otherwise, the return value is NULL, indicating that the error/status box could not be created or updated with new information.

Comments

The error/status listbox is created if no text has previously been added or displayed.

Text is added to the Engine Manager error/status listbox regardless of whether the listbox is

visible or not. Text is always added to the end of the current listbox contents, which are scrolled up, if necessary, to ensure the most recently added text is in view.

C/C++ Example

```
hwnd = DSAddEngMgrStatusLine("processing buffer-ready message");
if (hwnd != NULL) {

    SetMenu(hwnd, hMenu);  // add application-specific menu to status box
}
```

See Also

DSHideEngMgrStatusWindow

DSShowEngMgrStatusWindow

Related Users Guide Topics

section 7.5.1 Internal Error Handling in Engine Manager

section 7.5.2 Engine Error Handling

section 7.5.3 Error/Status Window in Engine Manager

DSEngineOpen

HENGINE DSEngineOpen (pszEnginePath, pszWaveformPath)

```
LPCSTR  pszEnginePath;      // path where engine is located
LPCSTR  pszWaveformPath;    // path of engine waveform files
```

The DSEngineOpen function opens and initializes the specified DSP Engine, and returns an engine handle which can be used in numerous other Engine Manager functions, as well as some Hardware Library and Signal Processing Library functions. Some pre-defined constants can be

specified in place of **szEnginePath**, as shown below:

Parameter	Description
pszEnginePath	pointer to null-terminated string containing partially- or fully-qualified path of DSP Engine to open; alternatively, the constants defined below can be used:

Value	Meaning
DS_EO_HSM	search for and open Hypersignal-Macro as DSP Engine
DS_EO_HSA	search for and open Hypersignal-Acoustic as DSP Engine
DS_EO_HSMNRT	search for and open Hypersignal-Macro NRT as DSP Engine
DS_EO_MAT	search for and open MATLAB as DSP Engine

pszWaveformPath	pointer to null-terminated string containing fully-qualified path location of waveform files which might be accessed by the DSP Engine

Returns

The return value is non-zero, representing a valid engine handle if successful. Otherwise, the return value is NULL, indicating that the engine could not be located (path was not valid), or the engine could not be initialized. For further error checking, use the DSGetEngMgrErrorStatus routine (see C/C++ source code example below).

Comments

A partially specified engine path will cause the DSEngineOpen function to assume an engine type and perform a rudimentary search. For example, specifying only "matlab" in the engine path will

cause the DSEngineOpen function to attempt to open :x:\matlab\bin\matlab.exe", where "x" is the drive location of the Engine Manager library.

The waveform path is not checked for valid waveform files, other than to ensure that the path actually exists.

The DSEngineOpen call opens DSP Engines such as Hypersignal-Macro series or MATLAB programs and places them in command mode, ready to accept instructions. DSP Engines are opened so as not to be visible (and conserve memory by not loading graphics drivers or other user-interface objects, if possible). Internally, DSP Engines are opened and initialized using either a Windows WinExec call or functions inside shell.dll. If the WinExec call is used, the "show state" parameter is set to SW_HIDE, and in the case of DOS engines, a .pif file may be associated with the engine.

For more information, see also Signalogic DSP Software Users Guide section 7.2.1, Opening DSP Engines.

C/C++ Example

```
HENGINE hEngine;

hEngine = DSEngineOpen(szEnginePath, szWaveformPath);
if (!hEngine) {
    nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}
```

See Also

DSEngineClose

DSGetEngineUsage

DSSetEngineWaveformPath

DSSetEngineTemplatePath

DSExecEngineFunction

Related Users Guide Topics

section 7.2 Controlling DSP Engines

section 7.2.1 Opening DSP Engines

section 7.2.2 Sending DSP Engine Commands

section 7.3 Executing Functions in DSP Engines

DSEngineClose

UINT DSEngineClose (hEngine)

```
HENGINE hEngine;        // engine handle
```

The DSEngineClose function closes the specified DSP Engine and invalidates the engine handle.

Parameter	Description
hEngine	handle to engine obtained from previous DSEngineOpen call

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid, or the engine could not be closed. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines

(see C/C++ source code example below).

Comments

The Engine Manager library is designed to close any engines still open when it is unloaded from the system. Unloading occurs only when all applications which implicitly or explicitly loaded Engine Manager have terminated. However, for each application it is always a good idea to call DSEngineClose for any currently active engine handles just prior to terminating. This avoids unnecessary consumption of system resources should Engine Manager not be unloaded immediately because other applications are still using it.

Also it is good programming practice to set the hEngine parameter to NULL after a successful DSEngineClose, to avoid the possibility of accidentally using an engine handle value in a Hardware Library call which might have been reassigned to another engine, in use by another application.

For more information details about obtaining engine handles using the DSEngineOpen call, see also Signalogic DSP Software Users Guide section 7.2.1, Opening DSP Engines.

C/C++ Example

```
if (!DSEngineClose(hEngine)) {
```

```
    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}
else hEngine = NULL;
```

See Also

DSEngineOpen

DSGetEngineUsage

DSSetEngineWaveformPath

DSSetEngineTemplatePath

DSExecEngineFunction

Related Users Guide Topics

section 7.2 Controlling DSP Engines

section 7.2.1 Opening DSP Engines

section 7.2.2 Sending DSP Engine Commands

section 7.3 Executing Functions in DSP Engines

DSExecEngineFunction

UINT DSExecEngineFunction (hEngine, pszCommandStr, uMode)

```
HENGINE hEngine;        // engine handle
LPCSTR pszCommandStr;   // command string of function to execute
UINT uMode;             // mode of execution
```

The DSExecEngineFunction call sends a text string containing an function, along with any associated input parameters, to the specified engine for execution. The command string must be null-

terminated and can be as long as needed.

Parameter	Description
hEngine	engine handle which was previously obtained via call to DSEngineOpen
pszCommandStr	pointer to null-terminated command string containing engine function to be executed, and any associated input parameters
uMode	mode in which the engine function should be executed; one of the possible pre-defined values listed below should be specified:

Value	Meaning
DS_EEF_ASYNC	execute engine function asynchronously with respect to calling application (calling application continues to execute simultaneously); this is the default used if uMode is left NULL
DS_EEF_SYNC	execute engine function synchronously with respect to calling application (calling application waits for engine function to complete)

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid, or the command could not be executed. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines (see C/C++ source code example below).

Comments

DSExecEngineFunction is a general purpose function which allows DSP Engine functions to be invoked from user-defined calling application programs. This facility

is not to be confused with DSP Engine functions, or combinations of functions, which are executed automatically and transparently as a result of standard API calls inside the DSPower-HWLib or DSPower-SPLib libraries.

The purpose of the DSExecEngineFunction call is to allow engine functions to be called directly, assuming knowledge of the syntax and parameters of the engine call.

Either synchronous or asynchronous operation can be specified using the uMode parameter. Synchronous operation means the calling application waits for the engine function to complete; asynchronous operation means the calling application continues to execute simultaneously.

C/C++ Example 1

Below is an example which evaluates a difference equation using Hypersignal-Macro software as the engine. The difference equation creates the output waveform file "sp_avg" from the input waveform file "speech":

```
HENGINE hEngine;
int nErrorType;

hEngine = DSEngineOpen(DS_EO_HSM, "c:\\hsmacro\\waveform");

if (!hEngine) {
     nErrorType = DSGetEngMgrErrorStatus();
     ...do error processing...
}
```

```
DSExecEngineFunction(hEngine, "f_diffeq(\"sp_avg\", \"y[n] = y[n] + (x1[n] –
x1[n-M])/Fr\", \"speech\", \"\", 0, 0, 0)", DS_EEF_ASYNC);
```

In the above example, the difference equation was executed asynchronously, meaning that the calling application continued to run at the same time. In such a case, the calling application can register a window using the DSRegisterEngineMsgWnd function to receive a "program done" message when the function is complete.

Note the use of "\"" to specify quote marks within the command string for string parameters required by the f_diffeq engine function. For a list of DSPower series software "f_xxxx" engine functions, see section 2.1 above, DSP Engine Functions.

C/C++ Example 2

The example below uses a combination of a DSPower engine function and a built-in MATLAB function to execute a simple FFT:

```
HENGINE hEngine;
int nErrorType;

hEngine = DSEngineOpen(DS_EO_MAT, "c:\\matlab\\waveform");

if (!hEngine) {
    nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}

DSExecEngineFunction(hEngine, "x = f_load(\"speech.wav\")", DS_EEF_SYNC);
DSExecEngineFunction(hEngine, "y = fft(x)", DS_EEF_ASYNC);
```

The above example shows the DSPower series software f_load

function being used to load the time domain waveform file "speech.wav". Note also this is done synchronously, meaning the calling application would wait for the engine to complete the f_load function. The second DSExecEngineFunction simply passes a command string invoking a MATLAB Signal Processing Toolbox function directly, allowing the calling application to continue executing by specifying the DS_EEF_ASYNC constant. In such a case, the calling application can register a window using the DSRegisterEngineMsgWnd function to receive a "program done" message when the engine function is complete.

C/C++ Example 3

The example below shows the f_fft DSPower function being used to accomplish the same result as shown in Example 2 above, but in one step:

```
HENGINE hEngine;
int nErrorType;

hEngine = DSEngineOpen(DS_EO_MAT, "c:\\matlab\\waveform");

if (!hEngine) {
    nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}

DSExecEngineFunction(hEngine, "y = f_fft(\"speech.wav\")", DS_EEF_ASYNC);
```

See Also

DSRegisterEngineMsgWnd

DSSendEngineCommand

DSSetEngineTemplatePath

DSAcquireWvfrmFile

DSGenerateWvfrmFile

WM_DSPENGINE_PROGSTATUS

Related Users Guide Topics

section 7.3 Executing Functions in DSP Engines

section 7.4 Registering Callback Messages from DSP Engines

Program Status Messages from DSP Engines (inside section 7.4)

section 7.2.2 Sending DSP Engine Commands

DSGetEngineErrorStatus

UINT DSGetEngineErrorStatus (hEngine)

HENGINE hEngine; // engine handle

The DSGetEngineErrorStatus function retrieves the current error status for the specified engine.

Parameter	Description
hEngine	handle to engine obtained from previous DSEngineOpen call

Returns

The return value indicates the current error condition for the specified engine, if one exists; otherwise, the return value is NULL. For DSPower-HWLib or DSPower-SPLib functions which require an engine handle and which return NULL, DSGetEngineErrorStatus should be checked first, followed by

DSGetEngMgrErrorStatus (see C/C++ source code example below).

Comments

The Engine Manager (enmgr.dll or enmgr32.dll) error handling scheme is straightforward. Most calls return a non-zero value on success, and a zero (NULL) on failure. When calls into Engine Manager fail, there are two levels of error possibilities: the library itself (or "base level"), and engine-related. Engine-related error conditions are returned by the DSGetEngineErrorStatus call, while base-level error conditions are returned by DSGetEngMgrErrorStatus.

A current error condition is maintained for each active DSP Engine. If an error condition occurs but is not retrieved, and further engine-related calls are made which cause *another* error condition, DSGetEngineErrorStatus returns only the most recent error condition.

Engine Manager also maintains a "status listbox" which can be made visible using the DSShowEngMgrStatusWindow call. This status window shows a history of error conditions from DSPower series library calls.

For more information, see Signalogic DSP Software Users Guide sections 7.5.1, Internal Error Handling in Engine Manager, 7.5.2, Engine Error Handling, and 7.5.3,

Error/Status Window in Engine Manager.

C/C++ Example

Below is an example showing error investigation in case of failure of a DSAssignBoard call, which is located inside Hardware Library:

```
int nErrorType;

if (!DSAssignBoard(hEngine, szBoardDesignator, uBusType, ulOBaseAddr,
uMemBaseAddr)) {

    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
```

Note that the DSGetEngineErrorStatus function should only be used with a valid engine handle. Otherwise, it is possible to incur an "invalid engine handle" error, which may overwrite a previously existing error condition. See **Comments** above.

See Also

DSGetEngMgrErrorStatus

DSGetHWMgrErrorStatus

DSGetHWLibErrorStatus

DSEngineOpen

Related Users Guide Topics

section 7.5 Error Handling in Engine Manager

section 7.5.1 Internal Error Handling in Engine Manager

section 7.5.2 Engine Error Handling

section 7.5.3 Error/Status Window in Engine Manager

DSGetEnginePath

UINT DSGetEnginePath (hEngine, pszEnginePath)

HENGINE hEngine; // engine handle

LPSTR pszEnginePath; // engine path to be filled in and returned

The DSGetEnginePath function retrieves the path for the specified DSP Engine and returns the value in pszEnginePath.

Parameter	Description
hEngine	handle to engine obtained from previous DSEngineOpen call
pszEnginePath	pointer to string variable which will be filled in with the specified engine path and returned as a null-terminated string

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines (see C/C++ source code example below).

Comments

The string variable pointed to by pszBoardDesignator is assumed to allow enough space for a return value (no space validation of any kind is performed). At least 144

characters is required (see C/C++ source code example below).

Essentially, the DSGetEnginePath function should return the same path used in the original DSEngineOpen call. The value might be different if a pre-defined constant or partial path was used in the DSEngineOpen call.

C/C++ Example

```
char szEnginePath[144];

if (!DSGetEnginePath(hEngine, szEnginePath)) {

    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}
```

See Also

DSEngineOpen

DSSetEngineWaveformPath

DSSetEngineTemplatePath

Related Users Guide Topics

section 7.2 Controlling DSP Engines

section 7.2.1 Opening DSP Engines

DSGetEngineState

UINT DSGetEngineState (hEngine)

```
HENGINE hEngine;        // engine handle
```

The DSGetEngineState function retrieves the current state of the specified DSP Engine.

Parameter	Description
hEngine	handle to engine obtained from previous DSEngineOpen call

Returns

The return value is NULL if the engine handle was invalid. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines (see C/C++ source code example below). Otherwise, the return value is as defined below:

Value	Meaning
DS_ES_RUNNING	engine is in running (busy) state
DS_ES_IDLE	engine is in idle state
DS_ES_STOPPENDING	engine is running, but a stop (shutdown) command is pending
DS_ES_STOPPED	engine is stopped (shutdown)

Comments

The DSGetEngineState call can be used to learn the current state for a specified DSP Engine. Note that state changes can be determined dynamically by using DSRegisterEngineMsgWnd to register a window to receive WM_DSPENGINE_ENGINESTATE messages.

C/C++ Example

```
char szText[50];
short int nErrorType;

switch (DSGetEngineState(hEngine)) {

    case DS_ES_RUNNING:
```

```
            _fstrcpy(szText, "Running");
            break;

        case DS_ES_IDLE:

            _fstrcpy(szText, "Idle");
            break;

        case DS_ES_STOPPENDING:

            _fstrcpy(szText, "Stop Pending");
            break;

        case DS_ES_STOPPED:

            _fstrcpy(szText, "Shutdown");
            break;

        default:

            _fstrcpy(szText, "Error in DSGetEngineState Call");
            nErrorType = DSGetEngineErrorStatus(hEngine);
            if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();

            // do error processing...

            break;
}
MessageBox(NULL, szText, "Current Engine State", MB_OK);
```

See Also

WM_DSPENGINE_ENGINESTATE

DSRegisterEngineMsgWnd

DSEngineOpen

DSEngineClose

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Status Messages from DSP Engines

section 7.2 Controlling DSP Engines

section 7.2.1 Opening DSP Engines

DSGetEngineUsage

UINT DSGetEngineUsage ( )

The DSGetEngineUsage function returns the number of currently opened DSP Engines by the calling application.

Parameter	Description
None	

Returns

The return value is zero if no engines are currently opened by the calling application. Otherwise, the return value is a positive value indicating the number of engines currently open.

Comments

After invoking a DSEngineClose function for each open engine, DSGetEngineUsage should return 0.

One possible use for this call is in case of catastrophic engine failure, when it may be important to learn if one or more engines has been forced to close due to internal Engine Manager error processing.

C/C++ Example

```
UINT uNumEngines;
char szText[30];

uNumEngines = DSGetEngineUsage();
itoa(uNumEngines, szText, 10);
MessageBox(NULL, szText, "Number of open engines", MB_OK);
```

See Also

DSEngineOpen

DSEngineClose

Related Users Guide Topics

section 7.2 Controlling DSP Engines

section 7.2.1 Opening DSP Engines

DSGetEngMgrErrorStatus

UINT DSGetEngMgrErrorStatus ( )

The DSGetEngMgrErrorStatus function retrieves the current base-level error status for the Engine Manager library(enmgr.dll or enmgr32.dll).

Parameter	Description
None	

Returns

The return value indicates the current Engine Manager library error condition, if one exists; otherwise, the return value is NULL. DSGetEngineErrorStatus should be checked first after DSPower-HWLib or DSPower-SPLib calls which require an engine handle and which return NULL, followed by DSGetEngMgrErrorStatus (see C/C++ source code example below).

Comments

The Engine Manager error handling scheme is straightforward. Most calls return a non-zero value on success, and a zero (NULL) on failure. When calls into Engine Manager fail, there are two levels of error possibilities: the library itself (or "base level"), and engine-related. Engine-related error conditions are returned by the DSGetEngineErrorStatus call, while base-level error conditions are returned by DSGetEngMgrErrorStatus.

A single, base-level current error condition variable is maintained internally in the Engine Manager library. If an error condition occurs but is not retrieved, and further calls are made which cause *another* base-level library error condition, DSGetEngMgrErrorStatus returns only the most recent error condition.

Engine Manager also maintains an "error/status listbox" which can be made visible using the DSShowEngMgrStatusWindow call. This status window shows a history of error conditions from all DSPower series library calls.

For more information, see Signalogic DSP Software Users Guide sections 7.5.1, Internal Error Handling in Engine Manager, 7.5.2, Engine Error Handling, and 7.5.3, Error/Status Window in Engine Manager.

C/C++ Example

Below is an example showing error investigation in case of failure of a DSAssignBoard call, which is located inside Hardware Library:

```
int nErrorType;

if (!DSAssignBoard(hEngine, szBoardDesignator, uBusType, uIOBaseAddr,
uMemBaseAddr)) {

    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
```

Note that the DSGetEngineErrorStatus function is used first, since a valid engine handle was involved in the function call which returned NULL. Otherwise, for calls such as DSEngineOpen which do not require an engine handle, DSGetEngMgrErrorStatus should be checked first. Note also that it is possible for error conditions to be overwritten if consecutive function calls are made without error-checking. See **Comments** above.

See Also

DSGetEngineErrorStatus

DSGetHWMgrErrorStatus

DSGetHWLibErrorStatus

DSEngineOpen

Related Users Guide Topics

section 7.5 Error Handling in Engine Manager

section 7.5.1 Internal Error Handling in Engine Manager

section 7.5.2 Engine Error Handling

section 7.5.3 Error/Status Window in Engine Manager

DSHideEngMgrStatusWindow

void DSHideEngMgrStatusWindow ( )

The DSHideEngMgrStatusWindow function hides the Engine Manager error/status listbox window, if not already hidden. The window is not destroyed, nor is its current contents and text display altered.

Parameter	Description
None	

Returns

No return value.

Comments

At least one call to DSAddEngMgrStatusLine must occur before the error/status listbox is created.

By default, the error/status listbox is not visible. C/C++, Visual Basic, MATLAB, and LabView programs using Engine Manager (part of DSPower-HWLib and DSPower-SPLib software) must call DSShowEngMgrStatusWindow at some point to make the listbox visible.

C/C++ Example

```
if (MessageBox(NULL, "hide error/status window?", "??", MB_YESNO) ==
IDYES) {
```

```
    DSHideEngMgrStatusWindow();
}
```

See Also

DSAddEngMgrStatusLine

DSShowEngMgrStatusWindow

Related Users Guide Topics

section 7.5.1 Internal Error Handling in Engine Manager

section 7.5.2 Engine Error Handling

section 7.5.3 Error/Status Window in Engine Manager

DSRegisterEngineMsgWnd

UINT DSRegisterEngineMsgWnd (hEngine, uMsgType, hwnd)

```
HENGINE hEngine;    // engine handle
UINT uMsgType;      // message type(s)
HWND hwnd;          // handle of window to register
```

The DSRegisterEngineMsgWnd function registers a window or control handle to receive one or more types of callback messages. A combination of messages can be specified using the pre-defined constants listed below.

Parameter	Description
hEngine	engine handle which was previously obtained via call to DSEngineOpen
uMsgType	one or message types for which hwnd should be registered, as defined below:

Value	Meaning
DS_REMW_SETENGINESTATUSMSG	receive engine status messages
DS_REMW_SETPROGSTATUSMSG	receive engine program and function status messages
DS_REMW_SETDSPDATARDYMSG	receive buffer ready and flag ready messages
DS_REMW_SETSTATISTICSMSG	receive statistic and parameter update messages
DS_REMW_SETBUFNUMMSG	receive buffer number update messages
DS_REMW_SETLEVELMSG	receive data level messages
DS_REMW_SETALLMSG	receive all types of callback messages

hwnd window or control handle to be registered

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid, the window handle was invalid, or one or more message types could not be registered. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines.

Comments

See section 6.1, Library Messages, for information about callback messages which can be received by application programs. The enmgr.h include file supplied with DSPower-HWLib and DSPower-SPLib software lists all callback message types.

C/C++ Example

```
if (!DSRegisterEngineMsgWnd(hEngine,
DS_REMW_SETENGINESTATUSMSG | DS_REMW_SETDSPDATARDYMSG,
hwnd)) {

    MessageBox(NULL, "error in DSRegisterEngineMsgWnd", "", MB_OK);
}
```

See Also

DSUnregisterEngineMsgWnd

DSWaitForBuffer

DSWaitForFlag

DSExecEngineFunction

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Status Messages from DSP Engines

Program Status Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

Update Messages from DSP Engines

DSSendEngineCommand

UINT DSSendEngineCommand (hEngine,
nCommand)
uMode)

```
HENGINE hEngine;        // engine handle
short int nCommand;     // command to send
UINT uMode;             // mode of execution
```

The DSSendEngineCommand function sends commands directly to the specified DSP Engine. Commands can be pre-defined

constants defined below, or ASCII or user-defined values as needed.

Parameter	Description
hEngine	engine handle which was previously obtained via call to DSEngineOpen
nCommand	command to send to engine; possible pre-defined command values are listed below:

Value	Meaning
DS_SEC_ABORT	instruct engine to abort (quit)
DS_SEC_IDLE	instruct engine to enter idle mode
DS_SEC_PAUSE	suspend engine (only valid for DOS engines)
DS_SEC_RESUME	resume engine (only valid for DOS engines)

uMode — mode in which command should be sent; one of the possible pre-defined values listed below should be used:

Value	Meaning
DS_EEF_ASYNC	send command asynchronously with respect to calling application (calling application continues to execute simultaneously as engine executes command); this is the default used if uMode is left NULL
DS_EEF_SYNC	send command synchronously with respect to calling application (calling application waits for engine to complete command)

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid, or the command was not sent. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines.

Comments

Pre-defined nCommand constants described above are translated by Engine Manager into correct DSP Engine commands. ASCII printable or control characters are sent to the engine without modification. For example the calling application could send command strings to MATLAB by using the DSSendEngineCommand function in a loop which sent each character of the string, and replaced the terminating NULL with a 0x0a value (linefeed).

Pre-defined nCommand constants listed above are encoded as negative values; to ensure that a calling application does not sent commands which will be translated by Engine Manager, consult the enmgr.h include file.

Either synchronous or asynchronous command execution can be specified using the uMode parameter. Synchronous operation means the calling application waits for the engine to execute the command; asynchronous operation means the calling application continues to execute simultaneously.

NOTE: Currently, a uMode value of DS_EEF_ASYNC is only active for the DS_SEC_IDLE and DS_SEC_ABORT commands, which initialize a wait for a WM_DSPENGINE_PROGSTATUS message with DS_PS_PROGDONE parameter.

C/C++ Example 1

```
DSSendEngineCommand(hEngine, DS_SEC_IDLE);  // put engine in idle mode
```

C/C++ Example 2

```
char szCommand[100];
int i;

for (i=0; i<_fstrlen(szCommand); i++) {

    // send command text string

    DSSendEngineCommand(hEngine, szCommand[i], NULL);  // asynchronous
}
DSSendEngineCommand(hEngine, 0x0a, NULL);  // send Enter key
```

See Also

DSExecEngineFunction

WM_DSPENGINE_PROGSTATUS

Related Users Guide Topics

section 7.3 Executing Functions in DSP Engines

DSSetEngineTemplatePath

UINT DSSetEngineTemplatePath (hEngine, pszTemplatePath)

```
HENGINE hEngine;          // engine handle
LPCSTR pszTemplatePath;   // path for engine template files
```

The DSSetEngineTemplatePath function sets the path used by the specified engine for locating template files.

Parameter	Description
hEngine	engine handle; should be obtained using call to

	DSEngineOpen
pszTemplatePath	pointer to null-terminated string containing path of template files which may need to be accessed in creating engine programs for high-level functions

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid, or the path could not be set. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines.

Comments

Template files are source code files in the native language of the DSP Engine, and are used as the basis for creating executable engine programs. Depending on the DSP Engine, high-level functions such as DSAcquireWvfrmFile, DSGenerateWvfrmFile, and DSExecEngineFunction may be implemented using this technique.

C/C++ Example

```
DSSetEngineTemplatePath(hEngine, "c:\\dspower\\syslib");
```

See Also

DSAcquireWvfrmFile

DSGenerateWvfrmFile

DSExecEngineFunction

Related Users Guide Topics

section 4.5 Acquiring Data Using DSPower-HWLib Software

section 7.3 Executing Functions in DSP Engines

DSSetEngineWaveformPath

UINT DSSetEngineWaveformPath (hEngine, pszWaveformPath)

```
HENGINE hEngine;          // engine handle
LPCSTR pszWaveformPath;   // path for engine waveform files
```

The DSSetEngineWaveformPath function sets the path used by the specified engine for locating waveform files.

Parameter	Description
hEngine	engine handle; should be obtained using call to DSEngineOpen
pszWaveformPath	pointer to null-terminated string containing path of waveform files which might be accessed by the DSP Engine

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid, or the path could not be set. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines.

Comments

Waveform files can be in Hypersignal (.tim, .frq, etc.), MATLAB (.mat), Microsoft (.wav), or various acoustic/audio waveform file formats. For more information,

see chapter 4, Waveform File Formats.

C/C++ Example

```
DSSetEngineWaveformPath(hEngine, "c:\\hsmacro\\waveform");
```

See Also

DSAcquireWvfrmFile

DSGenerateWvfrmFile

DSExecEngineFunction

Related Users Guide Topics

section 4.5 Acquiring Data Using DSPower-HWLib Software

section 7.3 Executing Functions in DSP Engines

DSShowEngMgrStatusWindow

void DSShowEngMgrStatusWindow ( )

The DSShowEngMgrStatusWindow function makes the Engine Manager error/status listbox window visible, if not already visible. The window's current contents and text display are not altered.

Parameter	Description
None	

Returns

No return value.

Comments

At least one call to DSAddEngMgrStatusLine must occur before the error/status listbox is created.

By default, the error/status listbox is not visible. C/C++, Visual Basic, MATLAB, and LabView programs using Engine Manager (part of DSPower-HWLib and DSPower-SPLib software) must call DSShowEngMgrStatusWindow at some point to make the listbox visible.

C/C++ Example

```
DSAddEngMgrStatusLine("add this line before showing window");
DSShowEngMgrStatusWindow();
```

See Also

DSAddEngMgrStatusLine

DSHideEngMgrStatusWindow

Related Users Guide Topics

section 7.5.1 Internal Error Handling in Engine Manager

section 7.5.2 Engine Error Handling

section 7.5.3 Error/Status Window in Engine Manager

DSUnregisterEngineMsgWnd

UINT DSUnregisterEngineMsgWnd (hEngine, uMsgType)

```
HENGINE hEngine;    // engine handle
UINT uMsgType;      // message type(s)
```

The DSUnregisterEngineMsgWnd function unregisters the current window(s) associated with one or more types of callback messages, after which the specified callback message types are no longer sent to the window(s). A combination of message types can be specified using the pre-defined constants listed below.

Parameter	Description
hEngine	engine handle which was previously obtained using call to DSEngineOpen
uMsgType	one or more message types for which the current registered window(s) should be unregistered, as defined below:

Value	Meaning
DS_REMW_SETENGINESTATUSMSG	unregister window currently receiving engine status messages
DS_REMW_SETPROGSTATUSMSG	unregister window currently receiving engine program and function status messages
DS_REMW_SETDSPDATARDYMSG	unregister window currently receiving buffer ready and flag ready messages
DS_REMW_SETSTATISTICSMSG	unregister window currently receiving statistic and parameter update messages
DS_REMW_SETBUFNUMMSG	unregister window currently receiving buffer number update messages
DS_REMW_SETLEVELMSG	unregister window currently receiving data level messages
DS_REMW_SETALLMSG	unregister all windows currently receiving any callback messages

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid or one or more message types could not be unregistered. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines.

Comments

See section 6.1, Library Messages, for information about callback messages which can be received by application programs. The enmgr.h include file supplied with DSPower-HWLib and DSPower-SPLib software lists all callback message types.

C/C++ Example

```
DSUnregisterEngineMsgWnd(hEngine, DS_REMW_SETENGINESTATUSMSG
| DS_REMW_SETDSPDATARDYMSG);
```

See Also

DSRegisterEngineMsgWnd

DSWaitForBuffer

DSWaitForFlag

DSExecEngineFunction

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Status Messages from DSP Engines

Program Status Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

2.2.2 Hardware Manager Functions

[@k functions, Hardware Manager, hwmgr.dll, hwmgr32.dll]

Below are listed functions in the Hardware Manager API. For information on how to include Hardware Manager (hwmgr.dll or hwmgr32.dll) in C/C++, Visual Basic, MATLAB, and LabView projects, see Signalogic DSP Software Users Guide section 8.1, Hardware Manager Access.

Hardware Manager Functions

DSGetHWMgrCurVal
DSGetHWMgrEntry
DSSetHWMgrEntry
DSGetHWMgrEntryIndex
DSGetHWMgrErrorStatus
DSGetHWMgrNumEntries
DSGetHWMgrVar
DSReadHWSetupFile
DSSetHWMgrCurVal
DSSetHWMgrVar
DSShowHardwareSelectorDlg
DSWriteHWSetupFile

DSGetHWMgrCurVal

UINT (pszBoardDesignator)

DSGetHWMgrCurVal

```
LPSTR  pszBoardDesignator;        // pointer to board designator string
```

The DSGetHWMgrCurVal function retrieves the board designator for the current hardware value selected in the Hardware Selector.

Parameter	Description
pszBoardDesignator	pointer to string variable which will contain a null-terminated board designator string upon return

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the board designator string for the current hardware value in the Hardware Selector could not be retrieved. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

The string variable pointed to by pszBoardDesignator is assumed to allow enough space for a return value (no space validation of any kind is performed). At least 50 characters is suggested, although board designators are normally no longer than 10-12 characters. For more information about board designators, see Board Designator Field in Hardware Setup File, in Signalogic DSP Software Users

Guide section 8.2.2, DSP/Analog Hardware Configuration.

For more information about setting the current hardware value (including the current Board Designator field) in the Hardware Selector, see Signalogic DSP Software Users Guide section 8.2.1, Selecting DSP/Analog Hardware Types.

For more information about hardware values, see Signalogic DSP Software Users Guide section 8.2.3, Hardware Values and Variables.

C/C++ Example

The example below shows a DSGetHWMgrCurVal function call being used to retrieve the current Board Designator from the Hardware Selector, and then using this value to assign a board handle.

```
char szBoardDesignator[50];
int nErrorType;
HBOARD hBoard;

if (!DSGetHWMgrCurVal(szBoardDesignator)) {

    nErrorType = DSGetHWMgrErrorStatus();
    //  do error processing...
}
else hBoard = DSAssignBoard(hEngine, szBoardDesignator, NULL, NULL,
NULL);
```

See Also

DSSetHWMgrCurVal

DSAssignBoard

DSGetHWMgrEntryIndex

DSShowHardwareSelectorDlg

Related Users Guide Topics

8.2.1 Selecting DSP/Analog Hardware Types

8.2.3 Hardware Values and Variables

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

Board Designator, in section 8.2.2, DSP/Analog Hardware Configuration

Board Designator Field in Hardware Setup File, in section 8.2.2, DSP/Analog Hardware Configuration

DSGetHWMgrEntry

UINT DSGetHWMgrEntry (nIndex, pHWList)

```
short int nIndex;        // index into Hardware Setup File
PHWLIST pHWList;         // pointer to HWLISTENTRY structure
```

The DSGetHWMgrEntry function retrieves an entry from the Hardware Setup File, storing it in a HWLISTENTRY structure.

Parameter	Description
nIndex	index specifying which Hardware Setup File entry to retrieve
pHWList	pointer to HWLISTENTRY structure which will contain entry from Hardware Setup File

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating the Hardware Setup File index was out of range. For further error checking, use the DSGetHWMgrErrorStatus routine

(see C/C++ source code example below).

Comments

The Hardware Setup File is initially read into global memory (allocated by Hardware Manager) from disk file upon the first instance of a call into Hardware Manager requiring data DSP/analog hardware configuration data. This occurs automatically, and includes the DSShowHardwareSelectorDlg call.

It is possible to obtain DSP/analog hardware configuration data from Hardware Manager-allocated global memory by calling the DSGetHWMgrEntry function with a pointer to a HWLISTENTRY structure. If the information is modified and needs to be saved, DSSetHWMgrEntry and DSWriteHWSetupFile calls should be made. Note that if the Hardware Selector is invoked by the operator, and DSP/analog hardware configuration data subsequently changed followed by pushing the "OK" button, DSWriteHWSetupFile is executed automatically.

For more information, see Signalogic DSP Software Users Guide sections 8.3, Hardware Setup File, 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure (in section 8.3.2).

C/C++ Example

Below is an example of using the DSGetHWMgrEntryIndex and DSGetHWMgrEntry calls to obtain Hardware Setup File information:

```
int nErrorType, nIndex;
HWLISTENTRY HWListEntry;

_fmemset(&HWListEntry, 0, sizeof(HWLISTENTRY));
nIndex = DSGetHWMgrEntryIndex("IIC31");
if (!nIndex || !DSGetHWMgrEntry(&HWListEntry, nIndex)) {

    nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
```

Note that the HWListEntry structure is initialized to zero before using, causing most structure elements to be set their default state. Note also that DSGetHWMgrEntryIndex can return NULL if the Hardware Setup File entry specified by this function's pszBoardDesignator input parameter is not found.

See Also

HWLISTENTRY

DSGetHWMgrEntryIndex

DSGetHWMgrNumEntries

DSSetHWMgrEntry

DSReadHWSetupFile

DSWriteHWSetupFile

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

DSGetHWMgrEntryIndex

UINT DSGetHWMgrEntryIndex (pszBoardDesignator),

LPCSTR pszBoardDesignator; // pointer to board designator string

The DSGetHWMgrEntryIndex function retrieves the index of the Hardware Setup File entry matching the specified Board Designator.

Parameter	Description
pszBoardDesignator	pointer to null-terminated board designator string for which to find a Hardware Setup File index

Returns

The return value is NULL if the specified board designator string does not match any of the entries in the Hardware Setup File, or if there is a file-access problem. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

Index values returned by DSGetHWMgrEntryIndex can range from 0 to the current total number of Hardware Setup File entries – 1, inclusively; the upper value can be determined by calling DSGetHWMgrNumEntries.

The DSGetHWMgrEntryIndex function can be combined with DSGetHWMgrEntry and

DSSetHWMgrEntry to modify DSP/analog hardware configuration data in the Hardware Setup File. The Reference Guide descriptions for the latter two functions contain comments and detailed source code examples which provide more information about this process.

For more information about the arrangement, contents, and access of the Hardware Setup File, see Signalogic DSP Software Users Guide sections 8.3, Hardware Setup File, 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure (in section 8.3.2).

C/C++ Example

Below is an example of using the DSGetHWMgrEntryIndex function to look for the index of an entry in the Hardware Setup File matching a particular Board Designator string:

```
int nErrorType, nIndex;

nIndex = DSGetHWMgrEntryIndex("ARI3210");
if (!nIndex) {

    nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
```

Note that DSGetHWMgrEntryIndex returns NULL if the Hardware Setup File entry specified by the function's Board Designator string input parameter is not found. For more complete C/C++ source code examples, see Reference Guide descriptions for

DSGetHWMgrEntry and DSSetHWMgrEntry.

See Also

DSGetHWMgrNumEntries

DSGetHWMgrEntry

DSSetHWMgrEntry

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

Board Designator Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSGetHWMgrErrorStatus

UINT DSGetHWMgrErrorStatus ( )

The DSGetHWMgrErrorStatus function retrieves the current base-level error status for the Hardware Manager library (hwmgr.dll or hwmgr32.dll).

Parameter	Description
None	

Returns

The return value indicates the current Hardware Manager library error condition, if one exists; otherwise, the return value is NULL. See C/C++ source code example below.

Comments

The Hardware Manager error handling scheme is straightforward. Most calls return a non-zero value on success, and a zero (NULL) on failure. When calls into Hardware Manager fail, the DSGetHWMgrErrorStatus call should be invoked.

A single, base-level current error condition variable is maintained internally in the Hardware Manager library. If an error condition occurs but is not retrieved, and further calls are made which cause *another* base-level library error condition, DSGetHWMgrErrorStatus returns only the most recent error condition.

Note that Engine Manager maintains an "error/status listbox" which can be made visible using the DSShowEngMgrStatusWindow call. This status window shows a history of error conditions from all DSPower series library calls, including Hardware Manager.

For more information, see Signalogic DSP Software Users Guide sections 8.4, Error Handling in Hardware Manager, and 8.4.1, Internal Error Handling in Hardware Manager. See also section 7.5.3, Error/Status Window in Engine Manager.

C/C++ Example

Below is an example showing error investigation in case of failure of a DSGetHWMgrEntryIndex call:

```
int nErrorType, Index;

Index = DSGetHWMgrEntryIndex("IIC31");
if (!Index) {

    nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
```

See Also

DSGetEngMgrErrorStatus

DSGetEngineErrorStatus

DSGetHWLibErrorStatus

Related Users Guide Topics

section 8.4 Error Handling in Hardware Manager

section 8.4.1 Internal Error Handling in Hardware Manager

section 7.5.3 Error/Status Window in Engine Manager

DSGetHWMgrNumEntries

UINT DSGetHWMgrNumEntries ( ),

The DSGetHWMgrNumEntries function retrieves the total number of current entries in the Hardware Setup File.

Parameter	Description
None	

Returns

The return value is NULL if the Hardware Setup File has not yet been read or cannot be read into Hardware Manager memory, or

there are no valid entries in the file. Otherwise, the return value indicates the number of entries found in the Hardware Setup File. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

The DSGetHWMgrNumEntries function can be combined with DSGetHWMgrEntryIndex, DSGetHWMgrEntry, and DSSetHWMgrEntry to modify DSP/analog hardware configuration data in the Hardware Setup File. The Reference Guide descriptions for the latter two functions contain comments and detailed source code examples which provide more information about this process.

The DSGetHWMgrEntryIndex function can be used to retrieve the index of a Hardware Setup File entry matching a specified board designator string. This function returns values ranging from 0 to DSGetHWMgrNumEntries – 1, inclusively.

For more information about the arrangement, contents, and access of the Hardware Setup File, see Signalogic DSP Software Users Guide sections 8.3, Hardware Setup File, 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure (in section 8.3.2).

C/C++ Example

Below is an example of using the DSGetHWMgrNumEntries function to create a loop in which Hardware Setup File entries with certain configuration data are searched for:

```
int nErrorType, nNumEntries, i;
HWLISTENTRY HWListEntry;
WORD wMemBaseAddr;

nNumEntries = DSGetHWMgrNumEntries();
if (!nNumEntries) {

    nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
else for (i=0; i<nNumEntries; i++) {  // loop through all entries

    _fmemset(&HWListEntry, 0, sizeof(HWLISTENTRY));

    if (!DSGetHWMgrEntry(&HWListEntry, i)) {  // get the entry

        nErrorType = DSGetHWLibErrorStatus();
        ...do error processing...
        break; // exit loop
    }
    else {

        sscanf(HWListEntry.MemBaseAddr, %x, &wMemBaseAddr);

        if (wMemBaseAddr >= 0xD000  &&  wMemBaseAddr <= 0xEFFF) {

            // process memory-mapped hardware in region D000H - EFFFH
        }
    }
}
```

For additional C/C++ source code examples, see Reference Guide descriptions for DSGetHWMgrEntry and DSSetHWMgrEntry.

See Also

DSGetHWMgrEntry

DSSetHWMgrEntry

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

Board Designator Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSGetHWMgrVar

UINT DSGetHWMgrVar (wVarIndex, pszBoardDesignator)

```
WORD  wVarIndex;          // index of hardware variable
LPSTR  pszBoardDesignator; // pointer to board designator string
```

The DSGetHWMgrVar function retrieves the board designator string for the specified hardware variable in the Hardware Selector.

Parameter	Description
wVarIndex	index of hardware variable for which to retrieve board designator string; should be one of the pre-defined hardware variable constants listed below:

Value	Meaning
RTE	Real-Time Engine
SA	Spectrum Analyzer
ACC	DSP Accelerator
DTR	Digital Tape Recorder
ACQ	Data Acquisition (A/D conversion, waveform file record)
CSG	Continuous Signal Generation (periodic function generation and arbitrary signal synthesis)

SR	Stimulus & Response
PB	Playback (waveform file output)
DO	Digital Oscilloscope
UD1	User-Defined 1
UD2	User-Defined 2
UD3	User-Defined 3
UD4	User-Defined 4
UD5	User-Defined 5

pszBoardDesignator — pointer to string variable which will contain a null-terminated board designator string upon return

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating the hardware variable was out of range (greater than or equal to MAXBOARDVARS), or the board designator string could not be retrieved for the specified hardware variable in the Hardware Selector. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

Hardware variables appear in a subdialog box that is invoked from the Hardware Selector by pushing the button marked "Set Hardware Variables". Currently, this button is enabled only when the Hardware Selector is displayed from DSPower-Block Diagram software. However, hardware variables can still be set and retrieved at any time by user-defined programs using the DSGetHWMgrVar and

DSSetHWMgrVar functions in Hardware Manager.

The string variable pointed to by pszBoardDesignator is assumed to allow enough space for a return value (no space validation of any kind is performed). At least 50 characters is suggested, although board designators are normally no longer than 10-12 characters. For more information about board designators, see Board Designator Field in Hardware Setup File, in Signalogic DSP Software Users Guide section 8.3.1, Hardware Setup File Fields.

For more information about hardware variables, see Signalogic DSP Software Users Guide section 8.2.3, Hardware Values and Variables.

C/C++ Example

The example below shows a DSGetHWMgrVar function call being used to retrieve the board designator string for the data acquisition hardware variable from the Hardware Selector, and then using this value to assign a board handle.

```
char szBoardDesignator[50];
int nErrorType;
HBOARD hBoard;

if (!DSGetHWMgrVar(ACQ, szBoardDesignator)) {

    nErrorType = DSGetHWMgrErrorStatus();
    //  do error processing...
}
else hBoard = DSAssignBoard(hEngine, szBoardDesignator, NULL, NULL,
NULL);
```

See Also

DSSetHWMgrVar

DSGetHWMgrEntryIndex

DSShowHardwareSelectorDlg

DSGetHWMgrCurVal

DSSetHWMgrCurVal

DSAssignBoard

Related Users Guide Topics

section 8.2.3 Hardware Values and Variables

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

Board Designator, in section 8.2.2, DSP/Analog Hardware Configuration

Board Designator Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSGetHWSetupFile

UINT DSReadHWSetupFile (pszSetupFile)

```
LPSTR pszSetupFile;        // pointer to returned value of hardware setup file
```

The DSGetHWSetupFile function retrieves the current Hardware Setup File for the calling application. Note that no actual setup file access is made (see DSReadHWSetupFile description).

Parameter	Description
pszSetupFile	pointer to memory area sufficient to receive null-terminated Hardware Setup File string, which will contain full path information

Returns

The return value is NULL if the pszSetupFile parameter is NULL.

Comments

The current Hardware Setup File in use by a calling application can be specified by making DSSetHWSetupFile or DSReadHWSetupFile calls.

Currently, the default Hardware Setup File is named hwsetup.lst, and the default location of this file is the location of Hardware Manager (hwmgr.dll or hwmgr32.dll).

Hardware Manager keeps track of different setup files on an application-specific basis. Different applications may use different setup files or the same setup file. Also, different setup files may be open and updated at the same time. The current maximum number of applications which may be using Hardware Manager at one time is eight (8).

For more information, see Signalogic DSP Software Users Guide section 8.3, Hardware Setup File, section 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure in section 8.3.2.

See Also

DSReadHWSetupFile

DSSetHWSetupFile

DSWriteHWSetupFile

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

DSReadHWSetupFile

HGLOBAL DSReadHWSetupFile (pszSetupFile)

```
LPCSTR pszSetupFile;          // pointer to hardware setup file
```

The DSReadHWSetupFile function allocates global memory and reads HWLISTENTRY entries from the specified Hardware Setup File (containing DSP/analog hardware configuration data) into this memory A NULL value for pszSetupFile causes the default "hwsetup.lst" setup file, or previous setup file specified by DSSetHWSetupFile, to be read.

Parameter	Description
pszSetupFile	pointer to null-terminated Hardware Setup File string; this string should specify the location of the setup file, including path information if needed; without path information, the default location used by Hardware Manager (location of hwmgr.dll file or hwmgr32.dll) will be used

Returns

The return value is NULL if either the specified Hardware Setup File could not be located or a problem occurred allocating memory or reading the file. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example

below). Otherwise, the return value is the global memory handle to a series of HWLISTENTRY records containing DSP/analog hardware configuration data found in the specified setup file.

Comments

If no initial call is made to DSReadHWSetupFile, the default Hardware Setup File is initially read into global memory from disk file upon the first instance of a call into Hardware Manager requiring DSP/analog hardware configuration data. This occurs automatically, and includes the DSShowHardwareSelectorDlg call.

Memory is allocated as each new entry is read from the Hardware Setup File. If at some point, memory cannot be increased, NULL is returned and a subsequent call to DSGetHWMgrErrorStatus will return DSCOULDNOTINCREASEHWLISTMEM.

Currently, the default Hardware Setup File is named hwsetup.lst, and the default location of this file is the location of Hardware Manager (hwmgr.dll or hwmgr32.dll).

Hardware Manager keeps track of different setup files on an application-specific basis. Different applications may use different setup files or the same setup file. Also, different setup files may be open and updated at

the same time. The current maximum number of applications which may be using Hardware Manager at one time is eight (8).

It is possible to obtain and change DSP/analog hardware configuration data programmatically by calling the DSGetHWMgrEntry and DSSetHWMgrEntry functions with a pointer to a HWLISTENTRY structure. In this case, a DSWriteHWSetupFile call should be made if DSP/analog hardware configuration data has been changed, and needs to be saved. Alternatively, setup files may be altered directly using a standard text editor, outside of the Hardware Manager environment. Note that if the Hardware Selector is invoked by the operator, and DSP/analog hardware configuration data subsequently changed followed by pushing the "OK" button, DSWriteHWSetupFile is executed automatically.

If the DSReadHWSetupFile function is successful, the return value is the global memory handle to a series of HWLISTENTRY records containing DSP/analog hardware configuration data. The number of records can be determined by calling DSGetHWMgrNumEntries. The global memory handle is created using the GMEM_SHARED attribute.

NOTE: It is not recommended that calling applications access

HWLISTENTRY records directly using the global memory handle returned by DSReadHWSetupFile. Instead, the DSGetHWMgrEntry call should be used.

Note that the current hardware value setting, as well as hardware variable settings, are not stored in the Hardware Setup File. They are instead stored in a separate private configuration file maintained by Hardware Manager (hwmgr.ini), on a application-independent basis (separate sections inside the configuration file are used).

For more information, see Signalogic DSP Software Users Guide section 8.3, Hardware Setup File, section 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure in section 8.3.2.

C/C++ Example

The example below shows the DSReadHWSetupFile function being called:

```
HGLOBAL hHWList;
int nErrorType;
LPCSTR szSetupFile = NULL;

hHWList = DSReadHWSetupFile(szSetupFile);

if (!hHWList) {

    nErrorType = DSGetHWMgrErrorStatus();
    //  do error processing...
}
```

See Also

DSWriteHWSetupFile

DSGetHWMgrEntry

DSSetHWMgrEntry

DSGetHWMgrNumEntries

DSShowHardwareSelectorDlg

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

DSSetHWMgrCurVal

UINT DSSetHWMgrCurVal (pszBoardDesignator)

```
LPCSTR pszBoardDesignator;      // pointer to board designator string
```

The DSSetHWMgrCurVal function sets the current hardware value in the Hardware Selector to match the specified board designator.

Parameter	Description
pszBoardDesignator	pointer to null-terminated string containing board designator

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the board designator string was invalid or no match could be found in the current Hardware Setup File entries. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source example below).

Comments

For more information about the current hardware value in the Hardware Selector, see Signalogic DSP Software Users Guide section 8.2.1, Selecting DSP/Analog Hardware Types.

For more information about hardware values, see Signalogic DSP Software Users Guide section 8.2.3, Hardware Values and Variables.

For more information about board designators, see Board Designator Field in Hardware Setup File, in Signalogic DSP Software Users Guide section 8.3.1, Hardware Setup File Fields.

C/C++ Example

The example below shows a DSSetHWMgrCurVal function call being used to set the current board designator prior to displaying the Hardware Selector dialog box:

```
char szBoardDesignator[50], szUserBoardDesignator[50];
int nErrorType;

lstrcpy(szBoardDesignator, "TBS56-B");

if (!DSSetHWMgrCurVal(szBoardDesignator)) {

    nErrorType = DSGetHWMgrErrorStatus();
    //  do error processing...
}
else {

    DSShowHardwareSelectorDlg(hwnd, szUserBoardDesignator); // hwnd is
    owner

    if (lstrcmpi(szBoardDesignator, szUserBoardDesignator) != 0) {

        // user has made a different choice; process...
```

```
    }
}
```

Note that the DSShowHardwareSelectorDlg call also returns a board designator. In the above example, if the user did not specify a new choice while the Hardware Selector was displayed, or pushed the "Cancel" button, then szBoardDesignator and szUserBoardDesignator would end up with the same value.

See Also

DSGetHWMgrCurVal

DSShowHardwareSelectorDlg

DSGetHWMgrEntryIndex

Related Users Guide Topics

section 8.2.1 Selecting DSP/Analog Hardware Types

section 8.2.3 Hardware Values and Variables

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

Board Designator, in section 8.2.2, DSP/Analog Hardware Configuration

Board Designator Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSSetHWMgrEntry

```
UINT DSSetHWMgrEntry ( nIndex,
                       pHWList )

short int nIndex;      // index into Hardware Setup File
PHWLIST pHWList;       // pointer to HWLISTENTRY structure
```

The DSSetHWMgrEntry function sets an entry in the Hardware Setup

File, using information contained in a HWLISTENTRY structure.

Parameter	Description
nIndex	index specifying which Hardware Setup File entry to set
pHWList	pointer to HWLISTENTRY structure containing information with which to set the Hardware Setup File entry

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating the nIndex parameter was out of range (greater than the current value returned by DSGetHWMgrNumEntries). For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

If nIndex is equal to the value returned by DSGetHWMgrNumEntries (i.e. nIndex is one more than the index of the last existing entry), the entry specified by pHWList will be added, rather than replacing an existing entry.

The Hardware Setup File is initially read into global memory (allocated by Hardware Manager) from disk file upon the first instance of a call into Hardware Manager requiring data DSP/analog hardware configuration data. This occurs automatically, and includes the DSShowHardwareSelectorDlg call.

It is possible to obtain and set DSP/analog hardware configuration data from Hardware Manager-allocated global memory by calling the DSGetHWMgrEntry and DSSetHWMgrEntry functions with a pointer to a HWLISTENTRY structure. If the information is modified and needs to be saved, a DSWriteHWSetupFile call should be made. Note that if the Hardware Selector is invoked by the operator, and DSP/analog hardware configuration data subsequently changed followed by pushing the "OK" button, DSWriteHWSetupFile is executed automatically.

For more information, see Signalogic DSP Software Users Guide sections 8.3, Hardware Setup File, 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure (in section 8.3.2).

C/C++ Example

Below is an example of using the DSGetHWMgrEntry call to modify Hardware Setup File information, which has previously been retrieved using DSGetHWMgrEntryIndex and DSGetHWMgrEntry:

```
int nErrorType, nIndex;
HWLISTENTRY HWListEntry;

nIndex = DSGetHWMgrEntryIndex("IIC31");
if (!nIndex || !DSGetHWMgrEntry(&HWListEntry, nIndex)) {

    nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
else {
```

```
    // make changes to HWLISTENTRY information

    lstrcpy(HWListEntry.IOBaseAddr, "0x340");
    lstrcpy(HWListEntry.VoltageRanges, "1.4,1.8");
    HWListEntry.BusType = DS_AB_PCAT;

    if (!DSSetHWMgrEntry(&HWListEntry, nIndex)) {

        nErrorType = DSGetHWLibErrorStatus();
        ...do error processing...
    }
}
```

Note that the HWListEntry structure is initialized to zero before using, causing most structure elements to be set their default state. Note also that DSGetHWMgrEntryIndex can return NULL if the Hardware Setup File entry specified by the function's board designator string input parameter is not found.

See Also

HWLISTENTRY

DSGetHWMgrEntryIndex

DSGetHWMgrNumEntries

DSGetHWMgrEntry

DSReadHWSetupFile

DSWriteHWSetupFile

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

DSSetHWMgrVar

UINT DSSetHWMgrVar (wVarIndex, pszBoardDesignator)

```
WORD wVarIndex;          // index of hardware variable
LPCSTR pszBoardDesignator; // pointer to board designator string
```

The DSSetHWMgrVar function sets the board designator string for the specified hardware variable in the Hardware Selector.

Parameter	Description
wVarIndex	index of hardware variable for which to set board designator string; should be one of the pre-defined hardware variable constants listed below:

Value	Meaning
RTE	Real-Time Engine
SA	Spectrum Analyzer
ACC	DSP Accelerator
DTR	Digital Tape Recorder
ACQ	Data Acquisition (A/D conversion, waveform file record)
CSG	Continuous Signal Generation (periodic function generation and arbitrary signal synthesis)
SR	Stimulus & Response
PB	Playback (waveform file output)
DO	Digital Oscilloscope
UD1	User-Defined 1
UD2	User-Defined 2
UD3	User-Defined 3
UD4	User-Defined 4
UD5	User-Defined 5

pszBoardDesignator	pointer to string variable containing a null-terminated board designator string

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating the hardware variable was out of range (greater than or equal to MAXBOARDVARS), indicating that the Board Designator string was invalid or no match could be found in the current Hardware Setup File entries. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

Hardware variables appear in a subdialog box that is invoked from the Hardware Selector by pushing the button marked "Set Hardware Variables". Currently, this button is enabled only when the Hardware Selector is displayed from DSPower-Block Diagram software. However, hardware variables can still be set and retrieved at any time by user-defined programs using the DSGetHWMgrVar and DSSetHWMgrVar functions in Hardware Manager.

The string variable pointed to by pszBoardDesignator is assumed to allow enough space for a return value (no space validation of any kind is performed). At least 50 characters is suggested, although board designators are normally no longer than 10-12 characters. For more information about board designators, see Board Designator Field in Hardware Setup File, in

Signalogic DSP Software Users Guide section 8.3.1, Hardware Setup File Fields.

For more information about hardware variables, see Signalogic DSP Software Users Guide section 8.2.3, Hardware Values and Variables.

C/C++ Example

The example below shows a DSSetHWMgrVar function call being used to set the Board Designator string for the continuous signal generation hardware variable in the Hardware Selector:

```
int nErrorType;

if (!DSSetHWMgrVar(CSG, "NAT32C")) {

    nErrorType = DSGetHWMgrErrorStatus();
    //  do error processing...
}
```

See Also

DSGetHWMgrVar

DSShowHardwareSelectorDlg

DSGetHWMgrCurVal

DSSetHWMgrCurVal

DSGetHWMgrEntryIndex

Related Users Guide Topics

section 8.2.3 Hardware Values and Variables

Board Designator, in section 8.2.2, DSP/Analog Hardware Configuration

Board Designator Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSSetHWSetupFile

UINT DSReadHWSetupFile (pszSetupFile)

```
LPCSTR pszSetupFile;        // pointer to hardware setup file
```

The DSSetHWSetupFile function sets the current Hardware Setup File for the calling application. Note that no actual setup file access is made (see DSReadHWSetupFile description).

Parameter	Description
pszSetupFile	pointer to null-terminated Hardware Setup File string; this string should specify the location of the setup file, including path information if needed; without path information, the default location used by Hardware Manager (location of hwmgr.dll file or hwmgr32.dll) will be used

Returns

The return value is NULL if the pszSetupFile parameter is NULL.

Comments

The current Hardware Setup File in use by a calling application can also be specified by making a DSReadHWSetupFile call.

Currently, the default Hardware Setup File is named hwsetup.lst, and the default location of this file is the location of Hardware Manager (hwmgr.dll or hwmgr32.dll).

Hardware Manager keeps track of different setup files on an application-specific basis.

Different applications may use different setup files or the same setup file. Also, different setup files may be open and updated at the same time. The current maximum number of applications which may be using Hardware Manager at one time is eight (8).

For more information, see Signalogic DSP Software Users Guide section 8.3, Hardware Setup File, section 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure in section 8.3.2.

See Also

DSGetHWSetupFile

DSReadHWSetupFile

DSWriteHWSetupFile

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

DSShowHardwareSelectorDlg

long DSShowHardwareSelectorDlg (**hwnd**, **pszBoardDesignator**)

HWND hwnd; // handle of owner window

LPSTR pszBoardDesignator; // pointer to board designator string

The DSShowHardwareSelectorDlg function activates and displays the Hardware Selector dialog box.

Parameter	Description
hwnd	handle of owner window for Hardware Selector dialog box, if any
pszBoardDesignator	pointer to string variable which will contain a null-terminated board designator string upon return

Returns

The return value is −1 if the Hardware Selector cannot be displayed, or if a problem occurs reading the Hardware Setup File. Otherwise, the low word of the return value contains the result of a DialogBoxParam call, and the high word contains a constant indicating whether configuration data for one or more hardware values was changed, or any hardware variables were changed, or both.

Typically, the low word of the return value will contain the identifier of the button pushed to close the dialog box, usually IDOK or IDCANCEL. The high word of the return value will have bit 0 set if the configuration data for a hardware value was changed, and bit 1 set if any hardware variables were changed. For further error checking if a −1 value should be returned, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

If not already read, the Hardware Setup File is read into global memory (allocated by Hardware Manager) from disk file when the DSShowHardwareSelectorDlg call is invoked.. If DSP/analog hardware configuration data for one or more hardware values is changed by the operator while the Hardware Selector dialog box is displayed (and subsequently followed by pushing the "OK" button), a DSWriteHWSetupFile call is executed automatically before DSShowHardwareSelectorDlg returns.

It is also possible to obtain and set DSP/analog hardware configuration data from Hardware Manager-allocated global memory by calling the DSGetHWMgrEntry and DSSetHWMgrEntry functions with a pointer to a HWLISTENTRY structure. If the information is modified and needs to be saved, a DSWriteHWSetupFile call should also be made

For more information about the arrangement, contents, and means of access of the Hardware Setup File, see Signalogic DSP Software Users Guide sections 8.3, Hardware Setup File, 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure (in section 8.3.2).

The string variable pointed to by pszBoardDesignator is assumed to allow enough space for a return value (no space validation of any kind is performed). At least 50 characters is suggested, although

board designators are normally no longer than 10-12 characters. For more information about board designators, see Board Designator Field in Hardware Setup File, in Signalogic DSP Software Users Guide section 8.3.1, Hardware Setup File Fields.

The DSShowHardwareSelectorDlg invokes a DialogBoxParam call with "HardwareSetup" as the resource name, and pszBoardDesignator as the extra parameter information.

C/C++ Example

Below is an example showing the DSShowHardwareSelectorDlg call being used to prepare for a Hardware Library DSAssignBoard call:

```
char szBoardDesignator[50];
HENGINE hEngine;

if (DSShowHardwareSelectorDlg(NULL, szBoardDesignator) != IDCANCEL) {

    // assign board handle:  engine, bd designator, bus type, IO & mem base
    addrs
    hBoard = DSAssignBoard(hEngine, szBoardDesignator, NULL, NULL,
    NULL);

    // initialize the board; make sure it's installed and responded
    if (!DSInitBoard(hBoard)) {

        itoa(DSGetEngineErrorStatus(hEngine), tmpstr, 10);
        MessageBox(NULL, tmpstr, "Test Prog:  DSInitBoard failed", MB_OK);
    }
}
```

In the above example, DSShowHardwareSelectorDlg is used to obtain a valid board designator string for use in an DSAssignBoard call. In this case,

the software operator makes an onscreen choice using the hardware selection listbox inside the Hardware Selector. Note that the above example assumes that DSEngineOpen was called at some point to produce a valid engine handle for use in the DSAssignBoard call.

See Also

DSGetHWMgrEntry

DSSetHWMgrEntry

DSReadHWSetupFile

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

Board Designator Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSWriteHWSetupFile

UINT DSWriteHWSetupFile (hHWList)

```
HGLOBAL hHWList;          // handle to DSP/analog configuration data
                          inside previously opened Hardware Setup File
```

The DSWriteHWSetupFile function writes current HWLISTENTRY records (containing DSP/analog hardware configuration data) in the global memory area specified by hHWList to its associated Hardware Setup File (previously specified in calls to

DSReadHWSetupFile or DSSetSetupFile).

Parameter	Description
hHWList	handle to global memory area containing DSP/analog hardware configuration data, maintained as series of HWLISTENTRY records

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating a problem in accessing the global memory area or its associated Hardware Setup File. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

Note that if the hHWList parameter is given as NULL, then Hardware Manager determines the correct value assuming the calling application is the current Windows running task.

Currently, the default Hardware Setup File is named hwsetup.lst, and the default location of this file is the location of Hardware Manager (hwmgr.dll or hwmgr32.dll).

Hardware Manager keeps track of different setup files on an application-specific basis. Different applications may use different setup files or the same setup file. Also, different setup files may be open and updated at

the same time. The current maximum number of applications which may be using Hardware Manager at one time is eight (8).

It is possible to obtain and change DSP/analog hardware configuration data by calling the DSGetHWMgrEntry and DSSetHWMgrEntry functions with a pointer to a HWLISTENTRY structure. In this case, writing the Hardware Setup File using DSWriteHWSetupFile should be done if DSP/analog hardware configuration data has been changed, and needs to be saved. Alternatively, setup files may be altered directly using a standard text editor, outside of the Hardware Manager environment. Note that if the Hardware Selector is invoked by the operator, and DSP/analog hardware configuration data subsequently changed followed by pushing the "OK" button, a DSWriteHWSetupFile call is executed automatically.

The hHWList parameter, which may be a return value from a previous call to the DSReadHWSetupFile function is a global memory handle to a series of HWLISTENTRY records containing DSP/analog hardware configuration data. The number of records can be determined by calling DSGetHWMgrNumEntries. The global memory handle is created using the GMEM_SHARED attribute.

NOTE: It is not recommended that calling applications access HWLISTENTRY records directly using the global memory handle returned by DSReadHWSetupFile. Instead, the DSGetHWMgrEntry call should be used.

Note that the current hardware value setting, as well as hardware variable settings, are not stored in the Hardware Setup File. They are instead stored in a separate private configuration file maintained by Hardware Manager (hwmgr.ini), on an application-independent basis (separate sections inside the configuration file are used).

For more information, see Signalogic DSP Software Users Guide section 8.3, Hardware Setup File, section 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure in section 8.3.2.

C/C++ Example

The example below shows the DSWriteHWSetupFile function being called:

```
int nErrorType;

if (!DSWriteHWSetupFile(NULL)) {

    nErrorType = DSGetHWMgrErrorStatus();
    //  do error processing...
}
```

See Also

DSReadHWSetupFile

DSGetHWMgrEntry

DSSetHWMgrEntry

DSGetHWMgrNumEntries

DSShowHardwareSelectorDlg

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in section 8.3.2

2.2.3 Hardware Library Functions

[@k functions,, DSPower-HWLib, functions, Hardware Library, hwlib.dll, hwlib32.dll]

Below are listed functions accessible in the Hardware Library API. For information on how to include Hardware Library (hwlib.dll or hwlib32.dll) in C/C++, Visual Basic, MATLAB, and LabView projects, see Signalogic DSP Software Users Guide section 4.2, Creating Programs for DSPower-HWLib and DSPower-SPLib.

Hardware Library Functions

DSAcquireWvfrmFile

DSAssignBoard

DSCalcSampFreq

DSCancelWaitBuffer

DSCancelWaitFlag

DSDisableBoard

DSDSPToIEEE

DSFreeBoard
DSGenerateWvfrmFile
DSGetBoardBaseAddr
DSGetBoardBusType
DSGetBoardClass
DSGetBoardInfo
DSGetHVarMem
DSGetMem
DSGetMemArch
DSGetMemSize
DSGetWordLength
DSHoldBoard
DSHoldProcessor
DSInitBoard
DSIEEEToDSP
DSLoadFileBoard
DSLoadFileProcessor
DSPutHVarMem
DSPutMem
DSReadBoardReg
DSResetBoard
DSResetProcessor
DSRunBoard
DSRunProcessor
DSSetBoardBaseAddr
DSSetBoardBusType
DSWaitForBuffer
DSWaitForFlag
DSWriteBoardReg

DSAcquireWvfrmFile

UINT DSAcquireWvfrmFile (hBoard, pConversionInfo)

```
HBOARD hBoard;                  // DSP/analog hardware handle
PCONVERSIONINFO pConversionInfo; // pointer to CONVERSIONINFO struct
                                 containing data acquisition or stimulus &
                                 response information
```

The DSAcquireWvfrmFile function performs a data acquisition or stimulus & response operation, storing data from one or more analog input channels on the specified DSP/analog hardware continuously to waveform file.

Parameter	Description
hBoard	handle to DSP/analog hardware previously obtained from DSAssignBoard call
pConversionInfo	pointer to CONVERSIONINFO structure containing information about the data acquisition or stimulus & response operation to be performed

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating the operation failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

-note about failure: because this is asynchronous function, engine

problems (waveform file errors, compile errors, etc.) received via WM_DSPENGINE_PROGSTATUS message

C/C++ Example

```
HLOCAL hConvInfo;
PCONVERSIONINFO pConvInfo;

// create CONVERSIONINFO object memory; use zero-initialization to set most
parameters to default values

hConvInfo = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE |
LMEM_DISCARDABLE, sizeof(CONVERSIONINFO));

pConvInfo = (PCONVERSIONINFO)LocalLock(hConvInfo);

// file into which to acquire data (including path information)

lstrcpy(pConvInfo->szFilename, szWaveformFilename);

// setup for three (3) channels, init channel list (example is sequential, but could
be
// arbitrary channel order)

pConvInfo->NumChan = 3;          // three channels
for (j=0; j<pConvInfo->NumChan; j++) pConvInfo->ChanList[j] = j;

// setup up sampling rate, amount, gain, trigger information

pConvInfo->Fs =22050;  // 22.05 kHz is desired sampling rate
pConvInfo->NumSamples = 100000L;  // acquire 100,000 samples
lstrcpy(pConvInfo->szGainList, "1");  // see notes in CONVERSIONINFO

// start acquisition

DSAcquireWvfrmFile(hBoard, pConvInfo);

// free and delete hConvInfo memory

LocalUnlock(hConvInfo);
LocalFree(hConvInfo);
```

In the above example, numerous elements of the CONVERSIONINFO structure have been left at their default (or zero) values. For instance, no

trigger information has been set, digital offset and scale factors are not active, and the default break mode (store acquired data if user-interrupted early) is in effect.

Note also that the stored data file will actually represent 300,000 total samples, because three channels were specified.

See Also

DSGenerateWvfrmFile

WM_DSPENGINE_PROGSTATUS

DSAssignBoard

DSExecEngineFunction

Related Users Guide Topics

section 4.5, Acquiring Data Using DSPower-HWLib Software

Program Status Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 4.1.1, Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

section 7.3, Executing Functions in DSP Engines

DSAssignBoard

HENGINE DSAssignBoard (hEngine,
pszBoardDesignator,
uBusType,
wIOBaseAddr,
wMemBaseAddr)

```
HENGINE hEngine;          // engine handle
LPCSTR pszBoardDesignator; // pointer to board designator string
UINT uBusType;            // bus type
WORD wIOBaseAddr;         // I/O base address
```

WORD wMemBaseAddr; // memory base address (if any)

The DSAssignBoard function allocates and assigns a handle for the specified DSP/analog hardware type. Some important hardware configuration parameters can be initialized at the same time. The hardware type is specified using a board designator string.

The returned handle can be used in numerous other Hardware Library function calls.

Parameter	Description
hEngine	engine handle previously obtained from DSEngineOpen call
pszBoardDesignator	pointer to null-terminated board designator string
uBusType	bus type of specified DSP/analog hardware as defined by the values below; a NULL value indicates the default Bus Type in the Hardware Setup File should be used

Value	Meaning
DS_AB_PCXT	PC-XT bus type
DS_AB_PCAT	PC-AT bus type
DS_AB_ENABLE80X86INST	enable 80x86 instruction set operations

wIOBaseAddr	I/O base address of specified DSP/analog hardware; a NULL value indicates the default I/O Base Address in the Hardware Setup File should be used
wMemBaseAddr	memory base address, if applicable, of specified DSP/analog hardware; a NULL value indicates the default Memory Base Address in the Hardware Setup File should be used

Returns

The return value is a non-zero DSP/analog hardware handle if successful. Otherwise, the return value is NULL, indicating the DSAssignBoard call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

The pszBoardDesignator parameter is a string value which specifies the DSP/analog hardware type. The Hardware Setup File maintained by Hardware Manager contains the current list of valid board designators. Note that this value may also be conveniently chosen by an operator using the Hardware Selector dialog box. For more information about board designators, see Board Designator Field in Hardware Setup File, in Signalogic DSP Software Users Guide section 8.3.1, Hardware Setup File Fields.

NULL can be used for the uBusType, wIOBaseAddr, and wMemBaseAddr parameters to cause the current Bus Type, I/O Base Address, or Memory Base Address values in the Hardware Setup File to be substituted for these parameters. These values can also be changed at any time using the DSSetBoardBusType and DSSetBoardBaseAddr functions.

Most types of DSP/analog hardware do not require a memory

base address. See Related Users Guide Topics below for more information. Also, see Signalogic DSP Software System Guide section 4.3.2, Memory-Mapped Transfer Gotchas.

C/C++ Example

```
hBoard = DSAssignBoard(hEngine, szBoardDesignator, uBusType,
uIOBaseAddr, uMemBaseAddr);

if (hBoard != NULL) {

    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}

if (!DSInitBoard(hBoard)) {

    // display message to user that board is not installed correctly or not
    // responding
}
```

See Also

DSShowHardwareSelectorDlg

Hardware Selector

Hardware Setup File

DSSetBoardBusType

DSSetBoardBaseAddr

DSEngineOpen

Related Users Guide Topics

section 4.1.1, Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

section 8.2, Hardware Selector

section 8.3, Hardware Setup File

Board Designator Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Bus Type Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

I/O Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Memory Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSCalcSampFreq

DWORD DSCalcSampFreq (hBoard,
SampFreqRequested,
nNumChan,
pnChanList,
pSampFreqActual)

```
HBOARD  hBoard;              // DSP/analog hardware handle
float  SampFreqRequested;    // desired sampling frequency
short int  nNumChan;         // number of channels
short int far*  pnChanList;  // pointer to channel list
float far*  pSampFreqActual; // pointer to actual sampling frequency
```

Given a desired sampling frequency, the DSCalcSampFreq function calculates the actual sampling frequency, and corresponding mode or control word value, for the specified DSP/analog hardware. In cases where the hardware does not support, or only partially supports, simultaneous sampling, the number of channels and channel list can also be specified.

The DWORD return value represents a mode or control word which can be sent to the DSP code

for use in programming the onboard analog I/O clock generator.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
SampFreqRequested	desired sampling frequency (IEEE 32-bit floating-point value)
nNumChan	number of channels; used only for hardware with analog I/O sampling which is not fully simultaneous
pnChanList	pointer to channel list; must have as many elements as specified by nNumChan; used only for hardware with analog I/O sampling which is not fully simultaneous
pSampFreqActual	pointer to IEEE 32-bit floating-point value which will contain actual sampling rate upon return

Returns

The return value is a non-zero mode or control word suitable for use by DSP code executing on the DSP/analog hardware to program onboard analog I/O sampling frequency clock generators. Otherwise, the return value is NULL, indicating the DSAssignBoard call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

Most types of DSP/analog hardware are capable of sampling input and output data at a limited range and/or number of discrete rates. The range and number vary tremendously from one type of

hardware to another. One common example is hardware with sigma-delta converters offering 16 rates varying from 8 kHz to 48 kHz. Another common example is hardware with successive approximation converters offering 65535 different rates varying from 150 Hz to 200 kHz.

The nNumChan and pnChanList parameters are needed only for DSP/analog hardware with analog I/O sampling which is not fully simultaneous. Otherwise, these parameters should be set to NULL.

Examples where nNumChan and pnChanList are required include hardware with one or more multiplexed analog inputs or outputs and hardware with "channel scan" or "channel group" analog inputs or outputs. In these cases, the DSP Engine uses the nNumChan and pnChanList parameters to calculate a mode or control word reflecting an "aggregate sampling rate", which is some multiple of the requested sampling rate. Note that the pSampFreqActual return value does not reflect this aggregate rate, but always indicates the actual sampling frequency that will be in effect for each channel. pnChanList is not required if the hardware has *only one* multiplexed converter, channel group, etc., in which case the aggregate rate is simply the actual rate multiplied by the number of channels.

The DWORD return value is a mode or control register word which can typically be used by DSP code executing on the DSP/analog hardware, or by the DSP Engine to access a host PC clock-generator register. In almost all cases, there is usually one register on the DSP/analog hardware which is responsible for sampling frequency clock generation. This may be an onchip timer, an external timer device, a PAL or PEEL, a CODEC control register, etc. Having this value calculated by the DSP Engine provides the main advantage of informing the operator or user of differences between desired and actual sampling rates *a priori*.

When using Hypersignal DSP Source Code Interface software functions or operating modes, the return mode or control word value can be written to the DSP_FSMODE location, and the value pointed to by pSampFreqActual can be written to the DSP_FSVALUE location.

A successful DSInitBoard function must have been completed beforehand.

C/C++ Example

```
DWORD dwFsMode;
float ActualFs, DesiredFs;

DesiredFs  = 75000;  // desired sampling rate is 75 kHz
dwFsMode = DSCalcSampFreq(hBoard, DesiredFs, NULL, NULL, &ActualFs);

DSPutHVarMem(hBoard, DSP_FSMODE, dwFsMode);
DSPutHVarMem(hBoard, DSP_FSVALUE, ActualFs);
```

In the above example, the nNumChan and pnChanList parameters have been ignored; these are only required for some hardware types with non-simultaneous sampling (e.g. inputs which are “scanned”, or sampled on a rotating basis, which repeats).

See Also

DSGetMemSize

DSGetMemArch

DSGetWordLength

DSPutHVarMem

DSP_FSMODE

DSP_FSVALUE

DSGetBoardClass

Related Users Guide Topics

section 4.3.3, Hardware Interrogation

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSCancelWaitBuffer

UINT DSCancelWaitBuffer (hBoard, nBufferNum)

```
HBOARD hBoard;          // DSP/analog hardware handle
short int nBufferNum;   // buffer number
```

The DSCancelWaitBuffer function cancels a pending DSP Engine wait-for-buffer request with the specified DSP/analog hardware.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
nBufferNum	number of buffer to cancel wait request for

Returns

The return value TRUE if the wait-for-buffer request was canceled successfully. Otherwise, the return value is NULL, indicating the DSCancelWaitBuffer call failed.

Comments

If a wait-for-buffer request was not pending for the specified buffer number, the DSCancelWaitBuffer call still returns TRUE.

If the wait-for-buffer request was established using interrupt-driven notification (DS_WFB_POLLED for uNotifyType parameter in DSWaitForBuffer), the DSP-to-host PC interrupt is disabled.

C/C++ Example

See Also

DSWaitForBuffer

WM_DSPENGINE_BUFRDY

DSCancelWaitFlag

Related Users Guide Topics

Buffer Ready Messages from DSP Engines, in section 7.4 Registering Callback Messages from DSP Engines

DSCancelWaitFlag

UINT DSCancelWaitFlag (hBoard, nFlagNum)

```
HBOARD hBoard;          // DSP/analog hardware handle
short int nFlagNum;     // flag number
```

The DSCancelWaitFlag function cancels a pending DSP Engine wait-for-semaphore request with the specified DSP/analog hardware.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
nFlagNum	number of flag to cancel wait request

Returns

The return value TRUE if the wait-for-semaphore request was canceled successfully. Otherwise, the return value is NULL, indicating the DSCancelWaitFlag call failed.

Comments

If a wait-for-semaphore request was not pending for the specified flag number, the DSCancelWaitFlag call still returns TRUE.

If the wait-for-semaphore request was established using interrupt-driven notification (DS_WFB_POLLED for uNotifyType parameter in DSWaitForFlag), the DSP-to-host PC interrupt is disabled.

C/C++ Example

See Also

DSWaitForFlag

WM_DSPENGINE_FLGRDY

DSCancelWaitBuffer

Related Users Guide Topics

Semaphore Messages from DSP Engines, in section 7.4 Registering Callback Messages from DSP Engines

DSDisableBoard

UINT DSDisableBoard (hBoard)

```
HBOARD hBoard;          // DSP/analog hardware handle
```

The DSDisableBoard function disables the specified DSP/analog hardware, placing all DSP processors in reset and disabling other onboard resources which may be active.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call

Returns

The return value is non-zero if the DSP/analog hardware is disabled successfully. Otherwise, the return value is NULL, indicating the DSDisableBoard call failed.

Comments

DSDisableBoard is the antithesis of DSInitBoard. It places all DSP devices on the DSP/analog

hardware in reset, and disables other resources that may have been initialized in the course of other Hardware Library calls and/or subsequent DSP code operations. Examples include disabling host PC memory-mapped areas (if any), turning off sampling rate clock generators, disabling output drivers, etc.

After a DSDisableBoard call, a DSResetBoard call should be invoked prior to making further Hardware Library calls to the DSP/analog hardware.

NOTE: DSDisableBoard should not be called unless a successful DSInitBoard function has been completed previously.

C/C++ Example

```
DisableBoard(hBoard);  // shut-down board, including analog I/O
```

See Also

DSInitBoard

DSResetBoard

DSAssignBoard

Related Users Guide Topics

section 4.1.1, Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSDSPToIEEE

UINT (hBoard,

DSDSPToIEEE

uIEEEDataSize,
pSrcMem,
pDstMem,
dwCount)

```
HBOARD hBoard;          // DSP/analog hardware handle
UINT uIEEEDataSize;     // size of each IEEE floating-point data element
void far* pSrcMem;      // pointer to source buffer containing 32-bit floating-
                        point DSP data
void far* pDstMem;      // pointer to destination data buffer which will contain
                        IEEE floating-point data
DWORD dwCount;          // number of data elements to convert
```

The DSDSPToIEEE function converts DSP 32-bit floating-point data (in format corresponding to DSP devices associated with hBoard) to IEEE 32-bit floating-point data. Both source and destination data buffers are expected to be host PC memory defined by the calling application.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
uIEEEDataSize	size of individual IEEE floating-point data elements, as defined below:

Value	Meaning
DS_DTI_IEEESIZE32	32-bit IEEE data
DS_DTI_IEEESIZE64	64-bit IEEE data

pSrcMem	address in host PC memory of source buffer containing DSP 32-bit floating-point data
pDstMem	address in host PC memory of destination buffer which will contain IEEE 32-bit floating-point data
dwCount	number of data elements to convert

Returns

The return value is a TRUE if the conversion was completed successfully. Otherwise, the return value is NULL, indicating the DSDSPToIEEE call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

The purpose of the DSDSPToIEEE and DSIEEEToDSP functions is to convert calling application floating-point data to and from DSP floating-point data. These functions are most useful before and after DSGetMem and DSPutMem functions.

The DSDSPToIEEE and DSIEEEToDSP functions determine the DSP chip family type from the driver ID value associated with hBoard. This value is established from the Driver ID field in the Hardware Setup File when a DSAssignBoard call is made. Currently, for ***fixed-point DSP devices,*** the DSDSPToIEEE function treats DSP data as if it were in signed fractional form, which means that only IEEE data values in the range of ±1 will be generated and stored.

C/C++ Example

```
DWORD n,dwCount = 1024;
float dbuf[dwCount];
```

```
for n = 0; n<dwCount-1; n++) dbuf[n] = sin(2*pi*n*1000/10000);  // some IEEE
data

// convert data to native floating-point DSP format

DSIEEEToDSP(hBoard, DS_DTI_IEEESIZE32, &dbuf, &dbuf, dwCount);

// transfer converted data to memory on DSP board

 DSPutMem(hBoard, DS_GM_LINEAR_DATA, 0x1000,
   DSGetWordLength(hBoard), &dbuf, dwCount);
```

Notes about the above example

1) IEEE data is converted in-place in dbuf, which means trying to use this data again in the host PC program would produce incorrect results

2) The DSP memory address of 0x1000 was chosen arbitrarily; depending on the type of DSP/acquisition board referenced by hBoard, this location could be in onchip SRAM or offchip SRAM

3) The DSGetWordLength function is a convenient method to determine the natural word-size for a given board; in the above example, it is assumed the result will be 32-bits, because of the previous use of the DS_DTI_IEEESIZE32 value

See Also

DSIEEEToDSP

DSGetMem

DSPutMem

Related Users Guide Topics

section 4.3.4, DSP Memory Data Transfers

Example C/C++ Code: Processing Buffer Ready Messages, in section 4.2.4,
Example Code: DSPower-HWLib and DSPower-SPLib Software

DSFreeBoard

UINT DSFreeBoard (hBoard)

```
HBOARD hBoard;          // DSP/analog hardware handle
```

The DSFreeBoard function unallocates and frees the specified board handle.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call

Returns

The return value is non-zero if the DSP/analog hardware handle is freed successfully. Otherwise, the return value is NULL, indicating the DSFreeBoard call failed.

Comments

Further Hardware Library calls using a handle previously freed by DSFreeBoard will return NULL.

DSFreeBoard should be invoked for any active DSP/analog hardware handles before calling applications which are using Hardware Library terminate. It may also be necessary to choose a handle to free if a recent call to DSAssignBoard indicates that all handles are allocated. In this case, the DSAssignBoard result is NULL

and the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines return the DSALLBOARDHANDLESALLOCATED error condition.

It is always a good idea to unallocate DSP/analog hardware handles as soon as they are no longer needed. This is because there may be multiple applications using Hardware Manager, requiring more handles at one time than is readily apparent.

C/C++ Example

See Also

DSAssignBoard

Related Users Guide Topics

section 4.1.1, Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSGenerateWvfrmFile

UINT DSGenerateWvfrmFile (hBoard, pConversionInfo)

HBOARD hBoard;	// DSP/analog hardware handle
PCONVERSIONINFO pConversionInfo;	// pointer to CONVERSIONINFO struct containing waveform file playback or signal generation information

The DSGenerateWvfrmFile function performs a waveform playback or continuous signal generation operation, sending data

to one or more analog output channels on the specified DSP/analog hardware continuously from waveform file.

Parameter	Description
hBoard	handle to DSP/analog hardware previously obtained from DSAssignBoard call
pConversionInfo	pointer to CONVERSIONINFO structure containing information about the waveform file playback or continuous signal generation operation to be performed

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating the operation failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

-note about failure: because this is asynchronous function, engine problems (waveform file errors, compile errors, etc.) received via WM_DSPENGINE_PROGSTATUS message

C/C++ Example

See Also

DSAcquireWvfrmFile

WM_DSPENGINE_PROGSTATUS

DSAssignBoard

DSExecEngineFunction

Related Users Guide Topics

section 4.5, Acquiring Data Using DSPower-HWLib Software

Program Status Messages from DSP Engines, in section 7.4, Registering Callback Messages from DSP Engines

section 4.1.1, Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

section 7.3, Executing Functions in DSP Engines

DSGetBoardBaseAddr

WORD DSGetBoardBaseAddr (hBoard, uAddrType)

```
HBOARD hBoard;      // DSP/analog hardware handle
UINT uAddrType;     // type of base address to retrieve
```

The DSGetBoardBaseAddr function retrieves the current I/O or memory base address for the specified DSP/analog hardware.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
uAddrType	type of base address to retrieve; can be one of the values defined below:

Value	Meaning
DS_GBBA_IOADDR	retrieve I/O base address
DS_GBBA_MEMADDR	retrieve memory base address

Returns

The return value is the current I/O or memory base address for the DSP/analog hardware specified by hBoard. Otherwise, the return

value is NULL, indicating the DSGetBoardBaseAddr call failed.

Comments

The values returned by DSGetBoardBaseAddr are either those initialized in the original DSAssignBoard call which created hBoard or set subsequently using a DSSetBoardBaseAddr call.

Note that if either the uIOBaseAddr or uMemBaseAddr parameters in the original DSAssignBoard call were NULL, then these values are initialized to the current I/O Base Address and Memory Base Address values in the Hardware Setup File.

C/C++ Example

See Also

DSSetBoardBaseAddr

DSAssignBoard

Related Users Guide Topics

I/O Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Memory Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSGetBoardBusType

UINT DSGetBoardBusType (hBoard)

```
HBOARD hBoard;          // DSP/analog hardware handle
```

The DSGetBoardBusType function retrieves the current bus type for the specified DSP/analog hardware.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call

Returns

The return value is the current bus type for the DSP/analog hardware specified by hBoard, shown by the values defined below. Otherwise, the return value is NULL, indicating the DSGetBoardBusType call failed.

Value	Meaning
DS_AB_PCXT	PC-XT bus type
DS_AB_PCAT	PC-AT bus type
DS_AB_ENABLE80X86INST	enable 80x86 instruction set operations

Comments

Depending on the speed of the host PC, and the type of DSP/analog hardware, there may be cases where specifying a different bus type can make the difference between correct operation of the hardware, or improve performance. For example, there are some DSP/analog hardware types which do not operate reliably when 80x86 type "string move" instructions are used, in which case the DS_AB_ENABLE80X86INST bus type attribute shown above should be avoided. For complete bus type information for specific hardware,

see chapter 3 in the Hypersignal-Macro Series Hardware Reference Guide.

The value returned by DSGetBoardBusType is either that initialized in the original DSAssignBoard call which created hBoard or set subsequently using a DSSetBoardBusType call.

Note that if the uBusType parameter in the original DSAssignBoard call was NULL, then this values are initialized to the current Bus Type value in the Hardware Setup File.

C/C++ Example

See Also

DSSetBoardBusType

DSAssignBoard

Related Users Guide Topics

Bus Type Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSGetBoardClass

WORD DSGetBoardClass (hBoard)

```
HBOARD hBoard;          // DSP/analog hardware handle
```

The DSGetBoardClass function retrieves a classification value suitable for use in differentiating between hardware types by DSP source code executing on the

DSP/analog hardware specified by hBoard.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call

Returns

The return value is a classification value for the specified DSP/analog hardware. Since classification values may be zero, the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should always be checked after this call (see C/C++ source code example below).

Comments

The DSGetBoardClass function returns a hardware classification value suitable for use by DSP Source Code Interface software. This value is maintained by DSP Engines because DSP Source Code Interface software for one type of DSP chip family may support multiple types of DSP/analog hardware, each with different configuration and analog I/O driver characteristics. Classification values can be sent to executing DSP source code during or prior to run-time initialization.

The result of the DSGetBoardClass function can be sent directly to the Hypersignal DSP Source Code software DSP_BOARDCLASS and DSP_BOARDSUBCLASS

variables (see source code example below).

For a list of classification values, see section 3.2, Board Designator, Driver ID, and Classification Table.

A successful DSInitBoard function must have been completed beforehand.

C/C++ Example

See Also

DSCalcSampFreq

DSGetMemSize

DSGetMemArch

DSGetWordLength

DSPutHVarMem

DSP_BOARDCLASS

DSP_BOARDSUBCLASS

Related Users Guide Topics

section 4.3.3, Hardware Interrogation

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSGetBoardInfo

UINT DSGetBoardInfo (hBoard, uInfoType)

```
HBOARD hBoard;      // DSP/analog hardware handle
UINT uInfoType;     // type of information to retrieve
```

The DSGetBoardInfo function retrieves internal Hardware Library

information for the specified DSP/analog hardware.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
uInfoType	type of information to retrieve, as indicated by one of the values defined below:

Value	Meaning
DS_GBI_ENGINEHANDLE	engine handle used in original DSAssignBoard call
DS_GBI_DRIVERID	driver ID value
DS_GBI_CALLALLOWED	call groups allowed flag
DS_GBI_HWMGRENTRYINDEX	Hardware Setup File index

Returns

The return value is the specified type of information for the DSP/analog hardware indicated by hBoard. Since all types of return values except for engine handles could legitimately be zero, the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should always be checked after this call (see C/C++ source code example below).

Comments

For more information about driver IDs and the call groups allowed flag, see Signalogic DSP Software Users Guide section 8.3.2, Hardware Setup File Access; see also section 5.1.1 HWLISTENTRY Structure, which describes the structure used to contain individual Hardware Setup File entries.

When uInfoType is DS_GBI_HWMGRENTRYINDEX, the return value can be used directly in the DSGetHWMgrEntry and DSSetHWMgrEntry function calls in Hardware Manager.

C/C++ Example

See Also

DSAssignBoard

DSGetHWMgrEntry

HWLISTENTRY

DSCalcSampFreq

DSGetMemSize

DSGetMemArch

DSGetWordLength

DSGetBoardClass

Related Users Guide Topics

section 4.1.1, Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

HWLISTENTRY Structure, in section 8.3.2, Hardware Setup File Access

section 4.3.3, Hardware Interrogation

DSGetHVarMem

DWORD DSGetHVarMem (hBoard, dwHVarAddr)

```
HBOARD hBoard;        // DSP/analog hardware handle
DWORD dwHVarAddr;     // address of variable
```

The DSGetHVarMem function reads a data element from the specified Hypersignal DSP source

code variable in DSP program memory on the DSP/analog hardware indicated by hBoard.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
dwHVarAddr	address of Hypersignal DSP source code variable to read; for a list of possible variables, see section 3.3, Hypersignal DSP Source Code Variable List

Returns

The return value is the DSP source code variable if read successfully. Since variables can legitimately be zero, the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should always be checked (see C/C++ source code example below).

Comments

The DSGetHVarMem and DSPutHVarMem functions are specific functions designed to transfer data between calling applications and Hypersignal DSP Source Code Interface software which has been loaded or is executing on DSP/analog hardware. Hypersignal DSP Source Code Interface software supports a wide variety of DSP/analog hardware types, and contains several built-in "operating modes" which can be controlled from host PC applications.

For general information, see Signalogic DSP Software Users

Guide section 5.0.2, DSP Source Code Interfaces for Hypersignal.

Note that all Hypersignal DSP source code variables are maintained in DSP program memory. Although the dwHVarAddr parameter can be any value, it should be one of the documented Hypersignal DSP source code variables. For a list of Hypersignal DSP source code variables, see section 3.3, Hypersignal DSP Source Code Variable List. In cases where user-defined dwHVarAddr values are used, the actual program memory location accessed will not be these values. The actual address can be calculated as:

```
dwActualAddr = dwHVarAddr – DSP_BOARDCLASS + VarStartAddr
```

DSP_BOARDCLASS is a Hypersignal DSP source code variable, as explained above (it is the first entry in the variable list maintained by Hypersignal DSP Source Code Interface software). In Hypersignal DSP Source Code Interface software, VarStartAddr is specific for each DSP device family. Note also that even though DSGetHVarMem allows for 32-bit values to be read, some DSP devices have a 16-bit or 24-bit natural wordlength. In these cases, only the least significant wordlength bits are read. The DSGetWordLength function can be used to determine wordlength for specified types of DSP/analog hardware. Below is a table with

common VarStartAddr and wordlength values.

DSP Family	VarStartAddr	Wordlength
ADSP-21xx	0x1c	16
DSP32C	0x40†	32
DSP3210	0x40†	32
DSP5600x	0x40	24
DSP9600x	0x100	32
TMS320C25	0x42f	16
TMS320C3x	0x400†	32
TMS320C4x	0x400	32
TMS320C5x	0x42f	16

All values for VarStartAddr listed above are given as if DSP device program memory is accessible only as contiguous locations each of wordlength bits. Values marked with † indicate that the DSP device may actually be byte- or word-addressable, but only the wordlength value shown is used for DSGetHVarMem and DSPutHVarMem addressing.

C/C++ Example

Below is an example of reading a DSP source code variable:

```
DWORD dwTimDataBaseAddr;

dwTimDataBaseAddr = DSGetHVarMem(hBoard, DSP_TIMDATAADDR);

if (!dwTimDataBaseAddr) {

    HENGINE hEngine = DSGetBoardInfo(hBoard, DS_GBI_ENGINEHANDLE);
    int nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();

    if (nErrorType != NULL) {
```

```
        // do error processing...
    }
}
```

The example above reads the default time data base address used by native Hypersignal DSP Source Code Interface functions and instruments. This example assumes a DSLoadFileBoard or DSLoadFileProcessor calls has previously been invoked, which would cause Hypersignal DSP source code variables to be initialized, depending on the program file that was downloaded.

See Also

DSPutHVarMem

DSGetMem

DSPutMem

DSGetWordLength

DSDSPToIEEE

DSIEEEToDSP

Related Users Guide Topics

section 4.3.4, DSP Memory Data Transfers

section 5.0.1, DSP Source Code Interface Documentation

section 5.0.2, DSP Source Code Interfaces for Hypersignal

section 5.0.3, DSP Source Code Interfaces for Hypersignal in Engine Mode

DSGetMem

UINT DSGetMem (hBoard,
uMemType,
dwDSPMemAddr,
uBitWidth,

pMem,
dwCount)

```
HBOARD hBoard;          // DSP/analog hardware handle
UINT uMemType;          // DSP memory type
DWORD dwDSPMemAddr;     // DSP memory start address
UINT uBitWidth;         // data element width in DSP memory
void far* pMem;         // pointer to buffer to receive data
DWORD dwCount;          // number of data elements to transfer
```

The DSGetMem function transfers one or more data elements from DSP/analog hardware memory to host PC memory defined by the calling application.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
uMemType	type of DSP memory from which to transfer data elements, as defined below:

Value	Meaning
DS_GM_LINEAR_PROGRAM	linear program memory
DS_GM_LINEAR_DATA	linear data memory
DS_GM_LINEAR_DATA_RT	linear data memory fast transfer while DSP is in run state (valid for some DSPs only)
DS_GM_VECTOR_DATA_X	vector X: data memory
DS_GM_VECTOR_DATA_Y	vector Y: data memory

dwDSPMemAddr	starting address in DSP data memory for transfer
uBitWidth	data element width in DSP memory for transfers, as defined below:

Value	Meaning
DS_GM_SIZE8	DSP memory data element width is 1 byte
DS_GM_SIZE16	DSP memory data element

	width is WORD
DS_GM_SIZE24	DSP memory data element width is 3 bytes
DS_GM_SIZE32	DSP memory data element width is DWORD or float

pMem — pointer to calling application memory which will receive data elements

dwCount — number of data elements to transfer

Returns

The return value is a TRUE if the transfer was completed successfully. Otherwise, the return value is NULL, indicating the DSGetMem call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

The DSGetMem and DSPutMem functions are flexible, general-purpose functions intended to transfer data between host PC memory and onboard or onchip DSP/analog hardware memory. Because of the wide variety of DSP/analog hardware memory types, there are several items to keep in mind:

1) Not all DSP devices have onchip memory which is accessible from outside the chip, and of those that do, not all board designs "project" accessibility to the host PC interface. In these cases, using the DSGetMem and DSPutMem functions with onchip DSP memory addresses will produce undefined

results. Consult DSP device and DSP/analog hardware documentation.

2) For almost all types of DSP/analog hardware, not all data element sizes of uBitWidth above are supported. In most cases, DS_GM_SIZE8 is not supported. For all hardware types except those based on Motorola DSP5600x and NEC77220 the DS_GM_SIZE24 value is not supported. In cases where a uBitWidth value is not supported, the transfer defaults to the next higher data element size.

3) Currently, no dynamic data conversion between integer and floating-point or IEEE floating-point and native DSP floating-point formats is made by DSGetMem or DSPutMem. This may change in later versions of Hardware Library, as new uMemType options are added. Thus, for example, if a calling MATLAB application is transferring 64-bit floating-point data to a DSP program, the DSP program may need to "know" to convert this data to its native 32-bit floating-point format before processing. Note, however, that for 32-bit floating-point DSP devices which do not use IEEE format (which is the default format used in Borland, Microsoft, and WATCOM C/C++ compilers), the DSDSPToIEEE and DSIEEEToDSP routines are available for converting native DSP 32-bit floating-point data to and from IEEE 32-bit floating-point

format. Note also that these routines only convert data using calling application memory buffers, and do not involve DSP memory.

4) For 32-bit floating-point DSP devices which support byte- or word-addressable memory, addressing used by DSGetMem and DSPutMem appears as if DSP memory is accessible as contiguous locations each of uBitWidth bits. For example, if a DSP device supports byte-addressable memory, then using a uBitWidth value of DS_GM_SIZE16 would cause access to occur at the *byte address* of dwDSPDataAddr∗2, and a uBitWidth value of DS_GM_SIZE32 would cause access to occur at the byte address of dwDSPMemAddr∗4 (or the *word address* of dwDSPMemAddr∗2). Examples of byte-addressable DSP devices include the AT&T DSP32C and DSP3210, and Texas Instruments TMS320C32.

5) The DS_GM_LINEAR_DATA_RT uMemType is intended for transferring 16-bit analog I/O data to DSP programs as fast as possible while the DSP device is in the run state. Currently, using the DS_GM_LINEAR_DATA_RT attribute will make a difference only with Texas Instruments TMS320 series devices. With all other devices, results are the same as using DS_GM_LINEAR_DATA. This

attribute is provided because it turns out that data transfer for many TMS320 series based DSP/analog hardware designs is faster if only the high 16-bits of data memory are used when transferring data to DSP memory, or judicious use (or non-use) of the hold state built in to these devices is required. In all cases except for TMS320C25 devices, and situations where a Hypersignal DSP Source Code Interface operating mode is being used which expects high-word transfers, it is safest to use the DS_GM_LINEAR_DATA construct.

6) Currently, the DS_GM_VECTOR_DATA_X and DS_GM_VECTOR_DATA_Y uMemType attributes are only supported for Motorola DSP5600x and DSP9600x devices. Regardless of the DSP type, DS_GM_VECTOR_DATA_X will produce the same result as DS_GM_LINEAR_DATA.

7) Experience has shown that it would be difficult to specify a straightforward processor list parameter in DSP memory data transfer functions. For multiprocessor DSP/analog hardware designs, data memory is generally shared, and different processor data spaces are chosen using memory address values, which allows DSGetMem and DSPutMem to be used as is. In cases where data or program memory between processors is

physically separate, DSWriteBoardReg and DSReadBoardReg calls may need to be made to select a memory space before using data transfer functions.

C/C++ Example

The source code example below transfers 32-bit data from DSP data memory address 0xc0000 to a calling application data buffer created by the Windows GlobalAlloc function. Note that the actual type of data being transferred could be either 32-bit integer or 32-bit floating-point, although the example assumes the data is floating-point.

```
HGLOBAL hBuf;
float far* pBuf;
WORD wBuflen;
DWORD dwDSPBaseAddr;

dwDSPBaseAddr = 0xc0000;

hBuf = GlobalAlloc(GMEM_MOVEABLE, wBufLen*sizeof(float));
pBuf = (float far*)GlobalLock(hBuf);

UINT uStatus = DSGetMem(hBoard, DS_GM_LINEAR_DATA,
dwDSPBaseAddr, DS_GM_SIZE32, pBuf, wBuflen);

if (!uStatus) {

    HENGINE hEngine = DSGetBoardInfo(hBoard,
    DS_GBI_ENGINEHANDLE));
    int nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    itoa(nErrorType, tmpstr, 10);
    MessageBox(NULL, tmpstr, "DSGetMem error occurred", MB_OK);
}
GlobalUnlock(hBuf);  // release mem
```

Note in the above example that a 32-bit floating-point device is

assumed. The above example would not be recommended for a 16-bit or 24-bit fixed-point DSP device.

See Also

DSPutMem

DSGetHVarMem

DSPutHVarMem

DSWaitForBuffer

WM_DSPENGINE_BUFRDY

DSGetWordLength

DSDSPToIEEE

DSIEEEToDSP

Related Users Guide Topics

section 4.3.4, DSP Memory Data Transfers

Example C/C++ Code: Processing Buffer Ready Messages, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSGetMemArch

UINT DSGetMemArch (hBoard)

```
HBOARD hBoard;          // DSP/analog hardware handle
```

The DSGetMemArch function retrieves the memory architecture for the DSP/analog hardware specified by hBoard.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call

Returns

If the DSGetMemArch function is successful, the return value is the type of memory architecture used on the specified DSP/analog hardware, as defined by the values below. Otherwise, the return value is NULL and the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should be checked for error conditions.

Value	Meaning
DS_GMA_LINEAR	memory architecture is linear, with program and data memory in the same address space
DS_GMA_HARVARD	memory architecture is modified Harvard, with program and data memory in separate address spaces
DS_GMA_VECTOR	memory architecture is modified Harvard, except data memory is further subdivided into two (2) vector spaces

Comments

The DSGetMemArch function returns a memory architecture type value suitable for use in DSGetMem and DSPutMem functions (see source code example below).

Typical examples of DSP devices which have a linear memory architecture are AT&T DSP32C and DSP3210, and Texas Instruments TMS320C3x and TMS320C4x. Typical examples of devices which have a modified Harvard memory architecture are Texas Instruments TMS320C5x. Typical examples of devices which

have a vector memory architecture are Motorola DSP5600x.

A successful DSInitBoard function must have been completed beforehand.

C/C++ Example

See Also

DSPutMem

DSGetMem

DSCalcSampFreq

DSGetMemSize

DSGetWordLength

DSGetBoardClass

Related Users Guide Topics

section 4.3.3, Hardware Interrogation

Example C/C++ Code: Processing Buffer Ready Messages, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSGetMemSize

DWORD DSGetMemSize (hBoard)

```
HBOARD hBoard;          // DSP/analog hardware handle
WORD wProcessorNum;     // value specifying which processor's data
                        memory to size
```

The DSGetMemSize function retrieves the maximum data memory size for the processor specified by wProcessorNum on the DSP/analog hardware specified by hBoard.

Parameter	Description

hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
wProcessorNum	value containing processor list; each bit represents a processor, starting with bit 0 for processor 0; only 1 bit should be set

Returns

If the DSGetMemSize function is successful, the return value is the maximum data memory size available for use with the specified processor. Otherwise, the return value is NULL and the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should be checked for error conditions.

Comments

The DSGetMemSize function will return NULL in the following cases:

- the wProcessorNum parameter indicates more than processor 0, and the DSP/analog hardware has only one processor (or in general indicates a higher processor number than actually exists)
- the wProcessorNum parameter is zero
- the wProcessorNum parameter has more than one bit set

The DSGetMemSize function returns the maximum amount of

data memory detected for the processor specified by wProcessorNum. For most hardware types, the return value is in units of the native wordlength of the DSP device. Some examples include 32-bit words for TMS320C3x and TMS320C4x devices, 24-bit words for DSP5600x devices, and 16-bit words for ADSP-21xx and TMS320C5x devices. However, there are currently two (2) exceptions. The DSGetMemSize return value is in 16-bit words for DSP32C and DSP3210 devices, even though these devices are 32-bit floating-point. This is because these devices have true multiwidth access to onchip and external memory, and some DSP Engine drivers take advantage of this feature to store data (especially analog I/O data) in compact (16-bit integer) form when possible.

In cases where the DSP/analog hardware has separate program and data memory areas, the DSGetMemSize function does not measure program memory.

In some cases, the return value may either be somewhat less than or exceed the amount of installed onboard, external data memory. This may be because external memory addressing is contiguous with onchip memory, or overlaps onchip memory (creating an unusable region).

A successful DSInitBoard function must have been completed beforehand.

C/C++ Example

See Also

DSGetMemArch

DSGetWordLength

DSPutMem

DSGetMem

DSCalcSampFreq

DSGetBoardClass

Related Users Guide Topics

section 4.3.3, Hardware Interrogation

DSGetWordLength

WORD DSGetWordLength (hBoard)

```
HBOARD hBoard;          // DSP/analog hardware handle
```

The DSGetWordLength function retrieves the natural word length of the DSP device(s) on the DSP/analog hardware specified by hBoard.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call

Returns

If the DSGetWordLength function is successful, the return value is the

natural wordlength, as defined in the table below, of the DSP device(s) used on the DSP/analog hardware. Otherwise, the return value is NULL and the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should be checked for error conditions.

Value	Meaning
DS_GM_SIZE16	DSP device is 16-bit fixed-point
DS_GM_SIZE24	DSP device is 24-bit fixed-point
DS_GM_SIZE32	DSP device is 32-bit floating-point

Comments

Some typical examples of DSGetWordLength return values:

AT&T DSP 32C, DSP 3210

Analog Devices ADSP-21xx

Analog Devices ADSP-21xxx	
Motorola DSP5600x	
Motorola DSP9600x	

Texas Instruments TMS320C25, TMS320C5x

Texas Instruments TMS320C3x, TMS320C4x

Note that this function does not actually "touch" hardware. The Driver ID field in the Hardware Setup File is used to determine the DSP chip family.

C/C++ Example

```
wDSPWordLength = DSGetWordLength(hBoard);

if (wDSPWordLength == NULL) {
```

```
    hEngine = DSGetBoardInfo(hBoard, DS_GBI_ENGINEHANDLE);
    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    // do error processing....
}
```

See Also

DSGetMemSize

DSGetMemArch

DSPutMem

DSGetMem

DSCalcSampFreq

DSGetBoardClass

Related Users Guide Topics

section 4.3.3, Hardware Interrogation

DSHoldBoard

UINT DSHoldBoard (hBoard)

HBOARD hBoard; // DSP/analog hardware handle

The DSHoldBoard function places all DSP processors on the DSP/analog hardware specified by hBoard into the hold state, if the hardware and DSP devices support this state.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call

Returns

The return value is non-zero if all DSP processors were placed in the hold state successfully, or if the DSP/analog hardware and/or DSP devices do not support this mode. Otherwise, the return value is NULL, indicating the DSHoldBoard call failed.

Comments

Upon successful return, all DSP devices on the DSP/analog hardware will be in the hold state--as long as a hold state is supported by the DSP/analog hardware host PC interface and the DSP device itself. For example, most Texas Instruments TMS320 series devices can be placed in hold state; only some of the DSP/analog hardware types with these processors allow hold state to be enabled from the host PC. Other types of DSP devices do not offer an externally-selectable hold state.

All DSP devices are held simultaneously, if the DSP/analog hardware supports a simultaneous-write mode. For multiprocessor hardware, if only certain DSP processors need to be held, use the DSHoldProcessor function.

Note that it varies between hardware types whether data transfer operations such as DSGetMem and DSPutMem may be executed while DSP devices are in the hold state. See documentation for the DSP/analog hardware type for specific information.

C/C++ Example

See Also

DSHoldProcessor

DSResetBoard

DSResetProcessor

DSRunBoard

DSRunProcessor

Related Users Guide Topics

section 4.3.1, Hardware and Processor Control

DSHoldProcessor

UINT DSHoldProcessor (hBoard, wProcessorList)

```
HBOARD hBoard;          // DSP/analog hardware handle
WORD wProcessorList;    // value specifying which processors should be held
```

The DSHoldProcessor function places one or more processors on the DSP/analog hardware specified by hBoard into the hold state, if the hardware and DSP devices support this state. The wProcessorList parameter is used to indicate which processors should be reset.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
wProcessorList	value containing processor list; each bit represents a processor, starting with bit 0 for processor 0; a 1 causes the processor to be included, a 0 causes it to be unaffected

Returns

The return value is non-zero if the DSP processors specified by wProcessorList were held successfully, or if the DSP/analog hardware and/or DSP devices do not support this mode. Otherwise, the return value is NULL, indicating the DSHoldProcessor call failed.

Comments

The DSHoldProcessor function will return NULL in the following cases:

the wProcessorList parameter indicates more than processor 0, and the DSP/analog hardware has only one processor

the wProcessorList parameter is zero

the wProcessorList parameter has more bits set to one than there are processors on the DSP/analog hardware

Upon successful return, DSP devices on the DSP/analog hardware specified by wProcessorList will be in the hold state--as long as a hold state is supported by the DSP/analog hardware host PC interface and the DSP devices themselves. For example, most Texas Instruments TMS320 series devices can be

placed in hold state; only some of the DSP/analog hardware types with these processors allow hold state to be enabled from the host PC. Other types of DSP devices do not offer an externally-selectable hold state.

DSP devices specified by wProcessorList are held simultaneously, if the DSP/analog hardware supports a simultaneous-write mode. For multiprocessor hardware, if all DSP processors need to be held simultaneously, it is simpler to use the DSHoldBoard function.

Note that it varies between hardware types whether data transfer operations such as DSGetMem and DSPutMem may be executed while DSP devices are in the hold state. See documentation for the DSP/analog hardware type for specific information.

C/C++ Example

See Also

DSHoldBoard

DSRunProcessor

DSRunBoard

DSResetBoard

DSResetProcessor

Related Users Guide Topics

section 4.3.1, Hardware and Processor Control

DSInitBoard

UINT DSInitBoard (hBoard)

HBOARD hBoard; // DSP/analog hardware handle

The DSInitBoard function checks for the presence, and if found, initializes the specified DSP/analog hardware.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call

Returns

The return value is non-zero if the DSP/analog hardware is located and initialized successfully. Otherwise, the return value is NULL, indicating the DSInitBoard call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

Initialization of DSP/analog hardware is straightforward, and occurs in the following steps:

A short register or onboard memory check is made to verify that the hardware is installed correctly.

Any DSP devices on the hardware are placed in reset state.

For those types of hardware which require it, a resident "host interface" DSP code segment may be downloaded and allowed to run. This is necessary in cases where the design of the hardware and/or DSP device does not allow onboard memory to be accessed while the DSP device is held in reset.

Note that in all cases, if the DSInitBoard function is successful, further operations such as DSGetMem and DSPutMem can be executed at any time. *If not, then further operations will produce undefined results*. The one exception to this is attempting additional DSInitBoard calls after using DSSetBoardBaseAddr to alter the I/O Base Address and/or Memory Base Address values for the hardware.

C/C++ Example

See Also

DSAssignBoard

DSLoadFileBoard

Related Users Guide Topics

section 4.1.1, Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSIEEEToDSP

```
UINT DSIEEEToDSP ( hBoard,
                   uIEEEDataSize,
                   pSrcMem,
                   pDstMem,
                   dwCount )

HBOARD hBoard;          // DSP/analog hardware handle
UINT uIEEEDataSize;     // size of each IEEE floating-point data element
void far* pSrcMem;      // pointer to source buffer containing IEEE floating-point data
void far* pDstMem;      // pointer to destination buffer which will contain 32-bit floating-point DSP data
DWORD dwCount;          // number of data elements to convert
```

The DSIEEEToDSP function converts IEEE 32-bit floating-point data to DSP 32-bit floating-point data (in format corresponding to DSP devices associated with hBoard). Both source and destination data buffers are expected to be host PC memory defined by the calling application.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
uIEEEDataSize	individual size of IEEE floating-point data elements, as defined by values below:

Value	Meaning
DS_DTI_IEEESIZE32	32-bit IEEE data

DS_DTI_IEEESIZE64	64-bit IEEE data
pSrcMem	address in host PC memory of source buffer containing IEEE floating-point data
pDstMem	address in host PC memory of destination buffer which will contain DSP 32-bit floating-point data
dwCount	number of data elements to convert

Returns

The return value is a TRUE if the conversion was completed successfully. Otherwise, the return value is NULL, indicating the DSIEEEToDSP call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

The purpose of the DSDSPToIEEE and DSIEEEToDSP functions is to convert calling application floating-point data to and from DSP floating-point data. These functions are most useful before and after DSGetMem and DSPutMem functions.

The DSDSPToIEEE and DSIEEEToDSP functions determine the DSP chip family type from the driver ID value associated with hBoard. This value is established from the Driver ID field in the Hardware Setup File when a DSAssignBoard call is made. Currently, for ***fixed-point DSP devices,*** the DSIEEEToDSP function attempts to convert IEEE data to signed fractional notation,

which means that IEEE data values outside of the range ±1 will "wrap around".

C/C++ Example

```
DWORD n,dwCount = 1024;
float dbuf[dwCount];

// get native floating-point DSP data from memory on DSP board

 DSGetMem(hBoard, DS_GM_LINEAR_DATA, 0x1000,
   DSGetWordLength(hBoard), &dbuf, dwCount);

// convert data to IEEE floating-point format

DSDSPToIEEE(hBoard, DS_DTI_IEEESIZE32, &dbuf, &dbuf, dwCount);
```

Notes about the above example

1) native DSP format data is converted in-place in dbuf, which allows the data to be used subsequently in the host PC program by referencing dbuf; attempting to perform floating-point operations on dbuf data after the DSGetMem function but before the DSDSPToIEEE function would produce incorrect results

2) The DSP memory address of 0x1000 was chosen arbitrarily; depending on the type of DSP/acquisition board referenced by hBoard, this location could be in onchip SRAM or offchip SRAM

3) The DSGetWordLength function is a convenient method to determine the natural word-size for a given board; in the above example, it is assumed the result will be 32-bits, because of the

DS_DTI_IEEESIZE32 value used in the conversion

See Also

DSDSPToIEEE

DSGetMem

DSPutMem

Related Users Guide Topics

section 4.3.4, DSP Memory Data Transfers

Example C/C++ Code: Processing Buffer Ready Messages, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSLoadFileBoard

UINT DSLoadFileBoard (hBoard, pszCodeFile)

```
HBOARD hBoard;          // DSP/analog hardware handle
LPCSTR pszCodeFile;     // pointer to executable DSP code file string
```

The DSLoadFileBoard function downloads an executable DSP code file to DSP program memory on the specified DSP/analog hardware. If the hardware contains more than one DSP processor with separate program memory, all processors receive the same executable code file.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
pszCodeFile	pointer to null-terminated string containing pathname of the executable DSP code file to download

Returns

The return value is TRUE if the executable DSP code file is downloaded successfully. Otherwise, the return value is NULL, indicating the DSLoadFileBoard call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

Successful DSAssignBoard and DSInitBoard calls must have been completed before DSLoadFileBoard calls are attempted.

If the pszCodeFile parameter is NULL, the DSLoadFileBoard call will download the current DSP Program File entry in the Hardware Setup File for the corresponding DSP/analog hardware type. This value can be retrieved independently by a call to DSGetHWMgrEntry using a Hardware Setup File index. There are two ways to get this index:

call DSGetHWMgrEntryIndex with the original board designator string used in DSAssignBoard

call DSGetBoardInfo with the hBoard parameter (same as above) and DS_GBI_HWMGRE

NTRYINDEX as input

Using the resulting index in a DSGetHWMgrEntry call will then retrieve a HWLISTENTRY structure, in which the pszCodeFile element is the current executable code file.

DSP executable code files should be in the format produced by the DSP chip manufacturer's linker utility. Typically, but not in all cases, DSP chip linkers produce COFF (common object file format) files.

NOTE: All DSP devices on the DSP/analog hardware should be in reset state before the DSLoadFileBoard call is invoked. Reset state is in effect after DSInitBoard, DSResetBoard, DSResetProcessor, and DSDisableBoard calls. Otherwise, results are undefined.

C/C++ Example

```
// load specified DSP program (code) file (if board is multiple-processor, then all
// processors are loaded with same file; see DSLoadFileProcessor below for
// processor-specific program file download)

if (!DSLoadFileBoard(hBoard, “dspcode.out”)) {

    // try default code file instead

    if (!DSLoadFileBoard(hBoard, NULL)) {

        MessageBox(NULL, " ", "Problem loading DSP program file", MB_OK);
    }
}
```

See Also

DSLoadFileProcessor

DSAssignBoard

DSInitBoard

DSP Program File

Related Users Guide Topics

section 4.1.1, Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSP Program File Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSLoadFileProcessor

UINT DSLoadFileProcessor (hBoard,
pszCodeFile,
wProcessorList)

```
HBOARD hBoard;          // DSP/analog hardware handle
LPCSTR pszCodeFile;     // pointer to executable DSP code file string
WORD wProcessorList;    // value specifying which processors should
                        receive the DSP code file
```

The DSLoadFileProcessor function downloads an executable DSP code file to DSP program memory for one or more processors on the specified DSP/analog hardware. The wProcessorList parameter is used to indicate which processors should receive the executable DSP code file.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
pszCodeFile	pointer to null-terminated string containing pathname of

the executable DSP code file to download

wProcessorList — value containing processor list; each bit represents a processor, starting with bit 0 for processor 0; a 1 enables downloading, a 0 disables

Returns

The return value is TRUE if the executable DSP code file is downloaded successfully. Otherwise, the return value is NULL, indicating the DSLoadFileProcessor call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

Successful DSAssignBoard and DSInitBoard calls must have been completed before DSLoadFileProcessor calls are attempted.

The DSLoadFileProcessor function will return NULL in the following cases:

- the wProcessorList parameter indicates more than processor 0, and the DSP/analog hardware has only one processor
- the wProcessorList parameter is zero
- the wProcessorList parameter has more bits set to one than there are processors

on the DSP/analog hardware

If the pszCodeFile parameter is NULL, the DSLoadFileProcessor call will download the current DSP Program File entry in the Hardware Setup File for the corresponding DSP/analog hardware type. This value can be retrieved independently by a call to DSGetHWMgrEntry using a Hardware Setup File index. There are two ways to get this index:

call DSGetHWMgrEntryIndex with the original board designator string used in DSAssignBoard

call DSGetBoardInfo with the hBoard parameter (same as above) and DS_GBI_HWMGRENTRYINDEX as input

Using the resulting index in a DSGetHWMgrEntry call will then retrieve a HWLISTENTRY structure, in which the pszCodeFile element is the current executable code file.

DSP executable code files should be in the format produced by the DSP chip manufacturer's linker utility. Typically, but not in all cases, DSP chip linkers produce COFF (common object file format) files.

NOTE: All DSP devices on the DSP/analog hardware should be in reset state before the DSLoadFileBoard call is invoked. Reset state is in effect after DSInitBoard, DSResetBoard, DSResetProcessor, and DSDisableBoard calls. Otherwise, results are undefined.

C/C++ Example

```
// load specified DSP program (code) file (processor 0 only)

if (!DSLoadFileProcessor(hBoard, “dspcode.out”, 0x01)) {

    // try default code file (processors 0, 1, and 2)

    if (!DSLoadFileProcessor(hBoard, NULL, 0x07)) {

        MessageBox(NULL, " ", "Problem loading DSP program file", MB_OK);
    }
}
```

See Also

DSLoadFileBoard

DSAssignBoard

DSInitBoard

DSP Program File

Related Users Guide Topics

section 4.1.1, Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSP Program File Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSPutHVarMem

UINT (hBoard,

DSPutHVarMem

dwHVarAddr,
dwHVarData)

```
HBOARD  hBoard;        // DSP/analog hardware handle
DWORD  dwHVarAddr;     // address of variable
DWORD  dwHVarData;     // data to send
```

The DSPutHVarMem function writes a data element to the specified Hypersignal DSP source code variable in DSP program memory on the DSP/analog hardware indicated by hBoard.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
dwHVarAddr	address of Hypersignal DSP source code variable to be written; for a list of possible variables, see section 3.3, Hypersignal DSP Source Code Variable List
dwHVarData	data to write to Hypersignal DSP source code variable

Returns

The return value is a TRUE if the variable was written successfully. Otherwise, the return value is NULL, indicating the DSPutHVarMem call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

The DSGetHVarMem and DSPutHVarMem functions are specific functions designed to

transfer data between calling applications and Hypersignal DSP Source Code Interface software which has been loaded or is executing on DSP/analog hardware. Hypersignal DSP Source Code Interface software supports a wide variety of DSP/analog hardware types, and contains several built-in "operating modes" which can be controlled from host PC applications.

For general information, see Signalogic DSP Software Users Guide section 5.0.2, DSP Source Code Interfaces for Hypersignal.

Note that all Hypersignal DSP source code variables are maintained in DSP program memory. Although the dwHVarAddr parameter can be any value, it should be one of the documented Hypersignal DSP source code variables. For a list of Hypersignal DSP source code variables, see section 3.3, Hypersignal DSP Source Code Variable List. In cases where user-defined dwHVarAddr values are used, the actual program memory location accessed will not be these values. The actual address can be calculated as:

```
dwActualAddr = dwHVarAddr – DSP_BOARDCLASS + VarStartAddr
```

DSP_BOARDCLASS is a Hypersignal DSP source code variable, as explained above (it is the first entry in the variable list maintained by Hypersignal DSP Source Code Interface software). In Hypersignal DSP Source Code

Interface software, VarStartAddr is specific for each DSP device family. Note also that even though dwHVarData allows for 32-bit values to be written, some DSP devices have a 16-bit or 24-bit natural wordlength. In these cases, only the least significant wordlength bits are written. The DSGetWordLength function can be used to determine wordlength for specified types of DSP/analog hardware. Below is a table with common VarStartAddr and wordlength values.

DSP Family	VarStartAddr	Wordlength
ADSP-21xx	0x1c	16
DSP32C	0x40†	32
DSP3210	0x40†	32
DSP5600x	0x40	24
DSP9600x	0x100	32
TMS320C25	0x42f	16
TMS320C3x	0x400†	32
TMS320C4x	0x400	32
TMS320C5x	0x42f	16

All values for VarStartAddr listed above are given as if DSP device program memory is accessible only as contiguous locations each of wordlength bits. Values marked with † indicate that the DSP device may actually be byte- or word-addressable, but only the wordlength value shown is used for DSGetHVarMem and DSPutHVarMem addressing.

C/C++ Example

Below is a short excerpt from the dscope.cpp source code example file included with DSPower-HWLib software:

```
DWORD dwFsMode;
WORD wBoardClass;
float FsActual;

// interrogate engine for board type value
wBoardClass = DSGetBoardClass(hBoard);

// find samp rate ctrl. reg. value and actual rate (closest rate possible to desired)
// 22.05 kHz sampling rate, assume simultaneous sampling

dwFsMode = DSCalcSampFreq(hBoard, 22050.0, NULL, NULL, &FsActual);

// load DSP Program File (usually a COFF file produced by DSP chip linker)
if (!DSLoadFileProcessor(hBoard, NULL, 0x01)) {  // load default file (proc 0 only)

    MessageBox(NULL, " ", "Dscope Demo:  problem loading file", MB_OK);
    goto cleanup;
}

// send down Hypersignal DSP source code variables in anticipation of using a
DSP source code operating mode
...
...
DSPutHVarMem(hBoard, DSP_BOARDCLASS, wBoardClass & 0x00ff);
DSPutHVarMem(hBoard, DSP_BOARDSUBCLASS, wBoardClass >> 8);
...
DSPutHVarMem(hBoard, DSP_FSMODE, dwFsMode);
...
```

See Also

DSGetHVarMem

DSGetMem

DSPutMem

DSGetWordLength

DSDSPToIEEE

DSIEEEToDSP

Related Users Guide Topics

section 4.3.4, DSP Memory Data Transfers

section 5.0.1, DSP Source Code Interface Documentation

section 5.0.2, DSP Source Code Interfaces for Hypersignal

section 5.0.3, DSP Source Code Interfaces for Hypersignal in Engine Mode

DSPutMem

```
UINT DSPutMem ( hBoard,
                uMemType,
                dwDSPMemAddr,
                uBitWidth,
                pMem,
                dwCount )
```

```
HBOARD hBoard;          // DSP/analog hardware handle
UINT uMemType;          // DSP memory type
DWORD dwDSPMemAddr;     // DSP memory start address
UINT uBitWidth;         // data element width in DSP memory
void far* pMem;         // pointer to buffer to receive data
DWORD dwCount;          // number of data elements to transfer
```

The DSPutMem function transfers one or more data elements from host PC memory defined by the calling application to DSP/analog hardware memory.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
uMemType	type of DSP memory to which to transfer data elements, as defined below:

Value	Meaning
DS_GM_LINEAR_PROGRAM	linear program memory
DS_GM_LINEAR_DATA	linear data memory
DS_GM_LINEAR_DATA_RT	linear data memory fast

	transfer while DSP is in run state (valid for some DSPs only)
DS_GM_VECTOR_DATA_X	vector X: data memory
DS_GM_VECTOR_DATA_Y	vector Y: data memory

dwDSPMemAddr — starting address in DSP data memory for transfer

uBitWidth — data element width in DSP memory for transfers, as defined below:

Value	Meaning
DS_GM_SIZE8	DSP memory data element width is 1 byte
DS_GM_SIZE16	DSP memory data element width is WORD
DS_GM_SIZE24	DSP memory data element width is 3 bytes
DS_GM_SIZE32	DSP memory data element width is DWORD or float

pMem — pointer to calling application memory which contains data elements to transfer

dwCount — number of data elements to transfer

Returns

The return value is a TRUE if the transfer was completed successfully. Otherwise, the return value is NULL, indicating the DSPutMem call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

The DSGetMem and DSPutMem functions are flexible, general-purpose functions intended to transfer data between host PC memory and onboard or onchip DSP/analog hardware memory. Because of the wide variety of

DSP/analog hardware memory types, there are several items to keep in mind:

1) Not all DSP devices have onchip memory which is accessible from outside the chip, and of those that do, not all board designs "project" accessibility to the host PC interface. In these cases, using the DSGetMem and DSPutMem functions with onchip DSP memory addresses will produce undefined results. Consult DSP device and DSP/analog hardware documentation.

2) For almost all types of DSP/analog hardware, not all data element sizes of uBitWidth above are supported. In most cases, DS_GM_SIZE8 is not supported. For all hardware types except those based on Motorola DSP5600x and NEC77220 the DS_GM_SIZE24 value is not supported. In cases where a uBitWidth value is not supported, the transfer defaults to the next higher data element size.

3) Currently, no dynamic data conversion between integer and floating-point or IEEE floating-point and native DSP floating-point formats is made by DSGetMem or DSPutMem. This may change in later versions of Hardware Library, as new uMemType options are added. Thus, for example, if a calling MATLAB application is transferring 64-bit floating-point data to a DSP program, the DSP program may need to "know" to convert this data to its native 32-bit

floating-point format before processing. Note, however, that for 32-bit floating-point DSP devices which do not use IEEE format (which is the default format used in Borland, Microsoft, and WATCOM C/C++ compilers), the DSDSPToIEEE and DSIEEEToDSP routines are available for converting native DSP 32-bit floating-point data to and from IEEE 32-bit floating-point format. Note also that these routines only convert data using calling application memory buffers, and do not involve DSP memory.

4) For 32-bit floating-point DSP devices which support byte- or word-addressable memory, addressing used by DSGetMem and DSPutMem appears as if DSP memory is accessible as contiguous locations each of uBitWidth bits. For example, if a DSP device supports byte-addressable memory, then using a uBitWidth value of DS_GM_SIZE16 would cause access to occur at the *byte address* of dwDSPDataAddr∗2, and a uBitWidth value of DS_GM_SIZE32 would cause access to occur at the byte address of dwDSPMemAddr∗4 (or the *word address* of dwDSPMemAddr∗2). Examples of byte-addressable DSP devices include the AT&T DSP32C and DSP3210, and Texas Instruments TMS320C32.

5) The DS_GM_LINEAR_DATA_RT

uMemType is intended for transferring 16-bit analog I/O data to DSP programs as fast as possible while the DSP device is in the run state. Currently, using the DS_GM_LINEAR_DATA_RT attribute will make a difference only with Texas Instruments TMS320 series devices. With all other devices, results are the same as using DS_GM_LINEAR_DATA. This attribute is provided because it turns out that data transfer for many TMS320 series based DSP/analog hardware designs is faster if only the high 16-bits of data memory are used when transferring data to DSP memory, or judicious use (or non-use) of the hold state built in to these devices is required. In all cases except for TMS320C25 devices, and situations where a Hypersignal DSP Source Code Interface operating mode is being used which expects high-word transfers, it is safest to use the DS_GM_LINEAR_DATA construct.

6) Currently, the DS_GM_VECTOR_DATA_X and DS_GM_VECTOR_DATA_Y uMemType attributes are only supported for Motorola DSP5600x and DSP9600x devices. Regardless of the DSP type, DS_GM_VECTOR_DATA_X will produce the same result as DS_GM_LINEAR_DATA.

7) Experience has shown that it would be difficult to specify a

straightforward processor list parameter in DSP memory data transfer functions. For multiprocessor DSP/analog hardware designs, data memory is generally shared, and different processor data spaces are chosen using memory address values, which allows DSGetMem and DSPutMem to be used as is. In cases where data or program memory between processors is physically separate, DSWriteBoardReg and DSReadBoardReg calls may need to be made to select a memory space before using data transfer functions.

C/C++ Example

The source code example below transfers quantized sine wave (1000 kHz sine sampled at 100 kHz) data to DSP data memory address 0x800 from a calling application data buffer created by the Windows GlobalAlloc function. Note that the actual type of DSP memory receiving the data could be anything from 16-bit, 24-bit, or 32-bit data--the host buffer contains only quantized 16-bit values stored in 32-bit signed integers.

```
HGLOBAL hBuf;
long far* pBuf;
WORD wBuflen;
DWORD dwDSPBaseAddr;

dwDSPBaseAddr = 0x800;

hBuf = GlobalAlloc(GMEM_MOVEABLE, wBufLen*sizeof(long));
pBuf = (long far*)GlobalLock(hBuf);
```

```
for n = 0; n<wBufLen-1; n++) pBuf[n] = (long)32767*sin(2*pi*n*1000/10000);

UINT uStatus = DSPutMem(hBoard, DS_GM_LINEAR_DATA,
dwDSPBaseAddr, DSGetWordLength(hBoard), pBuf, wBuflen);

if (!uStatus) {

    HENGINE hEngine = DSGetBoardInfo(hBoard,
    DS_GBI_ENGINEHANDLE));
    int nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    itoa(nErrorType, tmpstr, 10);
    MessageBox(NULL, tmpstr, "DSPutMem error occurred", MB_OK);
}
GlobalUnlock(hBuf);  // release mem
```

Note in the above example that the DSGetWordLength function is used to transfer data using a data element width equal to the natural wordlength of the DSP device. This method is reliable mainly for integer data, and not floating-point data, because of the varying storage representations of floating-point (or fractional integer format) data used by popular DSPs.

See Also

DSGetMem

DSGetHVarMem

DSPutHVarMem

DSWaitForBuffer

WM_DSPENGINE_BUFRDY

DSGetWordLength

DSDSPToIEEE

DSIEEEToDSP

Related Users Guide Topics

section 4.3.4, DSP Memory Data Transfers

Example C/C++ Code: Processing Buffer Ready Messages, in section 4.2.4,
Example Code: DSPower-HWLib and DSPower-SPLib Software

DSReadBoardReg

WORD DSReadBoardReg (hBoard,
wRegOffset)

```
HBOARD hBoard;          // DSP/analog hardware handle
WORD wRegOffset;        // offset from register base address
```

The DSReadBoardReg function reads a value from the specified register on the DSP/analog hardware indicated by hBoard.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
wRegOffset	offset from I/O base address of DSP/analog hardware

Returns

The return value is the current value of specified register on the DSP/analog hardware, assuming it can be read. Since register values can legitimately be NULL, the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should always be checked (see C/C++ source code example below).

Comments

In cases where directly accessing DSP/analog hardware is unavoidable, the DSReadBoardReg and DSWriteBoardReg functions may prove convenient.

In cases where registers are write-only, the return result is likely to be 0xffff. This may also be the case if

a location is read which is actually not a board register. Consult the DSP/analog hardware documentation for detailed information about I/O address space registers the hardware may support.

C/C++ Example

```
wVal = DSReadBoardReg(hBoard, wOffset);

HENGINE hEngine = DSGetBoardInfo(hBoard, DS_GBI_ENGINEHANDLE);
int nErrorType = DSGetEngineErrorStatus(hEngine);
if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();

if (nErrorType != NULL) {

    // do error processing...
}
```

See Also

DSWriteBoardReg

DSSetBoardBaseAddr

DSAssignBoard

Related Users Guide Topics

I/O Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSResetBoard

UINT DSResetBoard (hBoard)

```
HBOARD hBoard;          // DSP/analog hardware handle
```

The DSResetBoard function places all DSP processors on the DSP/analog hardware specified by hBoard into the reset state.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call

Returns

The return value is non-zero if all DSP processors were reset successfully. Otherwise, the return value is NULL, indicating the DSResetBoard call failed.

Comments

Upon successful return, all DSP devices on the DSP/analog hardware will be in the reset state.

All DSP devices are reset simultaneously, if the DSP/analog hardware supports a simultaneous-write mode. For multiprocessor hardware, if only certain DSP processors need to be reset, use the DSResetProcessor function.

The DSP Engine is responsible for accounting for necessary delays or special timing due to reset generators or other reset ciruitry.

For a few types of hardware, the DSResetBoard call may actually return with DSP devices in the run state, in which case a resident "host interface" DSP code segment will be active. This is necessary in cases where the design of the hardware and/or DSP device does not allow onboard memory to be accessed while the DSP device(s) are held in reset.

Note that in all cases, if the DSResetBoard function is successful, further operations such as DSGetMem and DSPutMem can be executed at any time. *If not, then further operations will produce undefined results.*

NOTE: DSResetBoard should not be called unless a successful DSInitBoard function has been completed previously.

C/C++ Example

See Also

DSResetProcessor

DSRunBoard

DSRunProcessor

DSInitBoard

Related Users Guide Topics

section 4.3.1, Hardware and Processor Control

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSResetProcessor

UINT DSResetProcessor (hBoard, wProcessorList)

```
HBOARD hBoard;          // DSP/analog hardware handle
WORD wProcessorList;    // value specifying which processors should be reset
```

The DSResetProcessor function places one or more processors on the DSP/analog hardware specified

by hBoard into the reset state. The wProcessorList parameter is used to indicate which processors should be reset.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
wProcessorList	value containing processor list; each bit represents a processor, starting with bit 0 for processor 0; a 1 causes the processor to be included, a 0 causes it to be unaffected

Returns

The return value is non-zero if the DSP processors specified by wProcessorList were reset successfully. Otherwise, the return value is NULL, indicating the DSResetProcessor call failed.

Comments

The DSResetProcessor function will return NULL in the following cases:

the wProcessorList parameter indicates more than processor 0, and the DSP/analog hardware has only one processor

the wProcessorList parameter is zero

the wProcessorList parameter has more bits set to one than there are processors on the DSP/analog hardware

Upon successful return, DSP devices on the DSP/analog hardware specified by wProcessorList will be in the reset state.

DSP devices specified by wProcessorList are reset simultaneously, if the DSP/analog hardware supports a simultaneous-write mode. For multiprocessor hardware, if all DSP processors need to be reset simultaneously, it is simpler to use the DSResetBoard function.

The DSP Engine is responsible for ensuring that necessary delays due to reset generators or other reset timing are accounted for.

For a few types of hardware, the DSResetProcessor call may actually return with DSP devices in the run state, in which case a resident "host interface" DSP code segment will be active. This is necessary in cases where the design of the hardware and/or DSP device does not allow onboard memory to be accessed while the DSP device(s) are held in reset.

Note that in all cases, if the DSResetProcessor function is successful, further operations such as DSGetMem and DSPutMem can be executed at any time. *If not, then further operations will produce undefined results*.

NOTE: DSResetProcessor should not be called unless a successful

DSInitBoard function has been completed previously.

C/C++ Example

See Also

DSResetBoard

DSRunProcessor

DSRunBoard

DSInitBoard

Related Users Guide Topics

section 4.3.1, Hardware and Processor Control

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSRunBoard

UINT DSRunBoard (hBoard)

```
HBOARD hBoard;          // DSP/analog hardware handle
```

The DSRunBoard function places all DSP processors on the DSP/analog hardware specified by hBoard into the run state.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call

Returns

The return value is non-zero if all DSP processors were started successfully. Otherwise, the return value is NULL, indicating the DSRunBoard call failed.

Comments

Upon successful return, all DSP devices on the DSP/analog hardware will be in the run state.

All DSP devices are started simultaneously, if the DSP/analog hardware supports a simultaneous-write mode. For multiprocessor hardware, if only certain DSP processors need to be started, use the DSRunProcessor function.

Note that in all cases, if the DSRunBoard function is successful, data transfer operations such as DSGetMem and DSPutMem may still be executed at any time. *If not, then further operations will produce undefined results.*

C/C++ Example

See Also

DSRunProcessor

DSResetBoard

DSResetProcessor

DSInitBoard

DSDisableBoard

Related Users Guide Topics

section 4.3.1, Hardware and Processor Control

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSRunProcessor

UINT DSRunProcessor (hBoard, wProcessorList)

```
HBOARD hBoard;          // DSP/analog hardware handle
WORD wProcessorList;    // value specifying which processors should be
                        run
```

The DSRunProcessor function places one or more processors on the DSP/analog hardware specified by hBoard into the run state. The wProcessorList parameter is used to indicate which processors should be started.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
wProcessorList	value containing processor list; each bit represents a processor, starting with bit 0 for processor 0; a 1 causes the processor to be included, a 0 causes it to be unaffected

Returns

The return value is non-zero if the DSP processors specified by wProcessorList were started successfully. Otherwise, the return value is NULL, indicating the DSRunProcessor call failed.

Comments

The DSRunProcessor function will return NULL in the following cases:

the wProcessorList parameter indicates more than processor 0, and the DSP/analog hardware has only one processor

the wProcessorList parameter is zero

the wProcessorList parameter has more bits set to one than there are processors on the DSP/analog hardware

Upon successful return, DSP devices on the DSP/analog hardware specified by wProcessorList will be in the run state.

DSP devices specified by wProcessorList are started simultaneously, if the DSP/analog hardware supports a simultaneous-write mode. For multiprocessor hardware, if all DSP processors need to be started simultaneously, it is simpler to use the DSRunBoard function.

Note that in all cases, if the DSRunProcessor function is successful, data transfer operations such as DSGetMem and DSPutMem can be executed at any time. *If not, then further operations will produce undefined results*.

C/C++ Example

See Also

DSRunBoard

DSResetProcessor

DSResetBoard

DSInitBoard

DSDisableBoard

Related Users Guide Topics

section 4.3.1, Hardware and Processor Control

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSSetBoardBaseAddr

UINT DSSetBoardBaseAddr (hBoard, uAddrType, wBaseAddr)

```
HBOARD hBoard;      // DSP/analog hardware handle
UINT uAddrType;     // type of base address to set
WORD wBaseAddr;     // base address value
```

The DSGetBoardBaseAddr function sets the current I/O or memory base address for the specified DSP/analog hardware.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
uAddrType	type of base address to set; can be one of the values defined below:

Value	Meaning
DS_GBBA_IOADDR	retrieve I/O base address
DS_GBBA_MEMADDR	retrieve memory base address

wBaseAddr	value to which to set base address

Returns

The return value TRUE if the I/O or memory base address was set successfully for the DSP/analog hardware specified by hBoard. Otherwise, the return value is

NULL, indicating the DSSetBoardBaseAddr call failed.

Comments

DSSetBoardBaseAddr can be used to set base addresses for DSP/analog hardware at any time. Note these values were initialized in the original DSAssignBoard call which created hBoard. Note also that if either the uIOBaseAddr or uMemBaseAddr parameters in the original DSAssignBoard calls were NULL, then these values are initialized to the current I/O Base Address and Memory Base Address values in the Hardware Setup File.

The DSGetBoardBaseAddr call can be used to retrieve DSP/analog hardware base address values at any time. The values returned are either those initialized in the original DSAssignBoard call or set subsequently using a DSSetBoardBaseAddr call.

C/C++ Example

See Also

DSGetBoardBaseAddr

DSAssignBoard

Related Users Guide Topics

I/O Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Memory Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSSetBoardBusType

UINT DSSetBoardBusType (hBoard,
uBusType)

```
HBOARD hBoard;      // DSP/analog hardware handle
UINT uBusType;      // type of base address to set
```

The DSSetBoardBusType function sets the current bus type for the specified DSP/analog hardware.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
uBusType	bus type to set; can be one of the values defined below:

Value	Meaning
DS_AB_PCXT	PC-XT bus type
DS_AB_PCAT	PC-AT bus type
DS_AB_ENABLE80X86INST	enable 80x86 instruction set operations

Returns

The return value TRUE if the bus type was set successfully for the DSP/analog hardware specified by hBoard. Otherwise, the return value is NULL, indicating the DSSetBoardBusType call failed.

Comments

Depending on the speed of the host PC, and the type of DSP/analog hardware, there may be cases where specifying a different bus type can make the difference between correct operation of the hardware,

or improve performance. For example, there are some DSP/analog hardware types which do not operate reliably when 80x86 type "string move" instructions are used, in which case the DS_AB_ENABLE80X86INST bus type attribute shown above should be avoided. For complete bus type information for specific hardware, see chapter 3 in the Hypersignal-Macro Series Hardware Reference Guide.

DSSetBoardBusType can be used to set bus types for DSP/analog hardware at any time. Note this value is initialized in the original DSAssignBoard call which created hBoard. Note also that if either the uBusType parameter in the original DSAssignBoard calls was NULL, then this value is initialized to the current Bus Type value in the Hardware Setup File.

The DSGetBoardBusType call can be used to retrieve DSP/analog hardware bus type values at any time. The value returned is either that initialized in the original DSAssignBoard call or set subsequently using a DSSetBoardBusType call.

C/C++ Example

See Also

DSGetBoardBusType

DSAssignBoard

Related Users Guide Topics

Bus Type Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSWaitForBuffer

```
UINT DSWaitForBuffer ( hBoard,
                       nBufferNum,
                       dwFlagAddr,
                       uNotifyType )

HBOARD hBoard;          // DSP/analog hardware handle
short int nBufferNum;   // buffer number
DWORD dwFlagAddr;       // flag address in DSP memory
UINT uNotifyType;       // notification type
```

The DSWaitForBuffer function establishes a DSP Engine wait-for-buffer request with the specified DSP/analog hardware.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
nBufferNum	number of buffer to wait for
dwFlagAddr	for user-defined buffers only, address of buffer flag in DSP memory
uNotifyType	notification type; can be one of the values defined below:

Value	Meaning
DS_WFB_POLLED	wait request is polled mode
DS_WFB_INT	wait request is interrupt-driven

Returns

The return value TRUE if the wait-for-buffer request was established

successfully. Otherwise, the return value is NULL, indicating the DSWaitForBuffer call failed.

Comments

A wait-for-buffer request is established with a DSP Engine by calling DSWaitForBuffer. Below is a short source code example (see also C/C++ Example below):

```
DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETDSPDATARDYMSG, hwnd);
DSWaitForBuffer(hBoard, nBufferNum, dwFlagAddr, uNotifyType);
```

In the above calls, hwnd is first registered to receive buffer ready and semaphore ready messages from the DSP Engine; in this case, hwnd will receive WM_DSPENGINE_BUFRDY messages.

nBufferNum values 0 through 15 are reserved for standard DSP Engine high-level function "data nodes", which are pre-defined sources and sinks of analog input and output data. In these cases, dwFlagAddr and uNotifyType are ignored. For example, nBufferNum values of 0 and 1 indicate continuous analog input data, alternating between buffers 0 and 1. uNotifyType is ignored for reserved nBufferNum values because DSP Engines may use specialized methods of buffer-checking, which may involve hardware register bits or other optimized techniques (not necessarily DSP board memory). The concept behind reserved data nodes is that more than one calling application may register a window

handle to receive the same buffer ready messages.

Values for nBufferNum not equal to 0 through 15 can be used to set up user-defined buffer ready messages. In these cases, dwFlagAddr and uNotifyType must be specified.

If a uNotifyType value of DS_WFB_POLLED is specified, the DSP Engine will periodically check to see if the specified buffer is ready, and cause WM_DSPENGINE_BUFRDY messages to be sent to hwnd as needed. If a uNotifyType value of DS_WFB_INT is specified, the DSP Engine will set up and enable a host PC interrupt, and send a WM_DSPENGINE_BUFRDY message when the interrupt occurs.

For more detailed information, see Buffer Ready Messages from DSP Engines, in Signalogic DSP Software Users Guide section 7.4, Registering Callback Messages from DSP Engines.

C/C++ Example

Below is another source code example taken from dscope.cpp, showing partial code for creating a window and designating it to receive buffer ready messages:

```
// create window that can be used for callback messages
hwnd = CreateWindow(wndclass.lpszClassName,
        "Example DSPower-HWLib Test Program",
        WS_OVERLAPPEDWINDOW | WS_VSCROLL | WS_HSCROLL |
        WS_CLIPCHILDREN | WS_VISIBLE,
        rectScopePos.left,
        rectScopePos.top,
```

```
        nWidth,
        nHeight,
        NULL,
        NULL, // no menu
        hInstance,
        NULL);

ShowWindow(hwnd, nCmdShow);
UpdateWindow(hwnd);

// scope window is born; turn the board loose...
DSRunProcessor(hBoard, 0x01);  // processor 0 only

// register window to receive buffer ready messages, tell engine to wait for input
data

DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETDSPDATARDYMSG, hwnd);
DSWaitForBuffer(hBoard, 0, NULL, DS_WFB_POLLED);

// no choice now but to follow Alice...
while (GetMessage(&msg, NULL, 0, 0)) {

    TranslateMessage (&msg);
    DispatchMessage (&msg);
};
```

See Also

WM_DSPENGINE_BUFRDY

DSCancelWaitBuffer

DSWaitForFlag

DSGetMem

DSPutMem

Related Users Guide Topics

Buffer Ready Messages from DSP Engines, in section 7.4 Registering Callback Messages from DSP Engines

Example C/C++ Code: Processing Buffer Ready Messages, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSWaitForFlag

UINT DSWaitForFlag (hBoard,

nFlagNum,
dwFlagAddr,
uNotifyType)

```
HBOARD hBoard;          // DSP/analog hardware handle
short int nFlagNum;     // flag number
DWORD dwFlagAddr;       // flag address in DSP memory
UINT uNotifyType;       // notification type
```

The DSWaitForFlag function establishes a DSP Engine wait-for-semaphore request with the specified DSP/analog hardware.

See below for function parameter descriptions.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
nFlagNum	number of flag to wait for
dwFlagAddr	for user-defined semaphores only, address of flag in DSP memory
uNotifyType	notification type; can be one of the values defined below:

Value	Meaning
DS_WFB_POLLED	wait request is polled mode
DS_WFB_INT	wait request is interrupt-driven

Returns

The return value TRUE if the wait-for-flag request was established successfully. Otherwise, the return value is NULL, indicating the DSWaitForFlag call failed.

Comments

A wait-for-semaphore request is established with a DSP Engine by calling DSWaitForFlag. Below is a short source code example (see also C/C++ Example below):

```
DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETDSPDATARDYMSG, hwnd);
DSWaitForFlag(hBoard, nFlagNum, dwFlagAddr, uNotifyType);
```

In the above calls, hwnd is first registered to receive buffer ready and semaphore ready messages from the DSP Engine; in this case, hwnd will receive WM_DSPENGINE_FLGRDY messages.

nFlagNum values 0 through 15 are reserved for standard DSP Engine high-level function "semaphore nodes", which are pre-defined status indicators for standard types of analog input and output data and acceleration processing. In these cases, dwFlagAddr and uNotifyType are ignored. For example, an nFlagNum value of 0 indicates accelerated file-based time domain data processing. uNotifyType is ignored for reserved nFlagNum values because DSP Engines may use specialized methods of flag-checking, which may involve hardware register bits or other optimized technique (not necessarily DSP board memory). The concept behind reserved semaphore nodes is that more than one calling application may register a window handle to receive the same semaphore messages.

Values for nFlagNum not equal to 0 through 15 can be used to set up

user-defined semaphore messages. In these cases, dwFlagAddr and uNotifyType must be specified.

If a uNotifyType value of DS_WFB_POLLED is specified, the DSP Engine will periodically check to see if the specified semaphore is ready, and cause WM_DSPENGINE_FLGRDY messages to be sent to hwnd as needed. If a uNotifyType value of DS_WFB_INT is specified, the DSP Engine will set up and enable the next available DSP-to-host PC interrupt, and send a WM_DSPENGINE_FLGRDY message to hwnd when the interrupt occurs.

For more detailed information, see Semaphore Messages from DSP Engines, in Signalogic DSP Software Users Guide section 7.4, Registering Callback Messages from DSP Engines.

C/C++ Example

See Also

WM_DSPENGINE_FLGRDY

DSCancelWaitFlag

DSWaitForBuffer

DSGetMem

DSPutMem

Related Users Guide Topics

Semaphore Messages from DSP Engines, in section 7.4 Registering Callback Messages from DSP Engines

DSWriteBoardReg

```
WORD                    ( hBoard,
DSWriteBoardReg
                          wRegOffset,
                          wVal )

HBOARD hBoard;          // DSP/analog hardware handle
WORD wRegOffset;        // offset from register base address
WORD wVal;              // value to write
```

The DSWriteBoardReg function writes a value to the specified register on the DSP/analog hardware indicated by hBoard.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
wRegOffset	offset from I/O base address of DSP/analog hardware
wVal	value to write to DSP/analog hardware register

Returns

The return value is TRUE if the value was written to the DSP/analog hardware register successfully. Otherwise, the value is NULL, indicating the DSWriteBoardReg function failed, and the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should be checked (see C/C++ source code example below).

Comments

In cases where directly accessing DSP/analog hardware is unavoidable, the DSReadBoardReg

and DSWriteBoardReg functions may prove convenient.

C/C++ Example

```
if (!DSWriteBoardReg(hBoard, wRegOffset, 0x59a2)) {

    HENGINE hEngine = DSGetBoardInfo(hBoard, DS_GBI_ENGINEHANDLE);
    int nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();

    if (nErrorType != NULL) {

        // do error processing...
    }
}
```

See Also

DSReadBoardReg

DSSetBoardBaseAddr

DSAssignBoard

Related Users Guide Topics

I/O Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

2.2.4 Tool Library Functions

[@k functions, Tool Library, toollib.dll, tlib32.dll]

Below is a partial list of functions in Tool Library, which is a library of user-interface objects and controls used internally at Signalogic and by products such as DSPower-Block Diagram. Tool Library is also included with DSPower-HwLib examples and demos, to provide a minimal example user-interface across a wide variety of platforms, including

various C/C++ programming environments (e.g. Borland, Microsoft, and WATCOM) and other programming environments such as Visual Basic . For information on how to include Tool Library DLLs (toollib.dll or tlib32.dll) in C/C++ and Visual Basic projects, see Signalogic DSP Software Users Guide section 8.1, Hardware Manager Access. The examples in this section refer to Hardware Manager, but references to hwlib.h can be replaced by toollib.h or toollib.bas. In addition, some of the individual function descriptions below contain C/C++ source code examples showing how to use Tool Library.

Tool Library Functions

DSAddToolboxControl

DSCreateToolbox

DSDeleteToolbox

DSGetToolboxFlags

DSGetToolboxRect

DSGetToolLibErrorStatus

DSGetToolLibFlags

DSGetUserInput

DSInitToolLib

DSMarkApp

DSMessageBox

DSMoveToolbox

DSSendMessageControl

DSSendMessageToolboxControl

DSSetToolboxFlags

DSAddToolboxControl

```
HCONTROL DSAddToolboxControl    ( hToolbox,
                                  ID,
                                  ControlType,
                                  pszLabel,
                                  pszHelpText,
                                  pCtrlItemsUser )

HTOOLBOX hToolbox;              // toolbox handle
WORD ID;                        // control ID
DWORD ControlType;              // control types
LPCSTR pszLabel;                // pointer to text string
                                containing label of the control
LPCSTR pszHelpText;             // pointer to text string
                                containing fly-overr help text
                                for the control
PCTRLITEMS pCtrlItemsUser;      // pointer to optional
                                CTRLITEMS structure
```

The DSAddToolboxControl function adds user-interface objects and control to a previously created toolbox. The toolbox may be thought of as a "container", which automatically sizes and shapes depending on the controls added. Objects and controls include toolbar buttons, pop-up dialogs, edit fields, knobs, dials, meters, list boxes, etc.

See below for DSAddToolboxControl calling parameter descriptions.

Parameter	Description
hToolbox	handle to toolbox previously obtained by DSCreateToolbox function

ID	user-specified control ID which can be used to identify control in WM_COMMAND messages or Visual Basic events; also pre-defined constants are available which specify toolbar button types and other
ControlType	type of specified control as defined in toollib.h and toollib.bas; see DS_ATC_xxx constants
pszLabel	pointer to text string which contains button text (for non bitmap-buttons), label for combo/edit/radio group, bitmap handle or bitmap filename for toolbar button, etc.
pszHelpText	pointer to null-terminated text string which contains fly-over help text, if any; default value is NULL
pCtrlItemsUser	optional pointer to CTRLITEMS structure which contains combo box or listbox items, pop-up dialog items, etc; default value is NULL

Returns

The return value is the handle to the new toolbox control, if successful. Otherwise, the return value is NULL, indicating that the control could not be created.

For further error checking, use the DSGetToolLibErrorStatus routine.

Comments

There are many pre-defined toolbox control and object types available in Tool Library. These are designated as DS_ATC_xxx, where "xxx" indicates the type of object.

For toolbar buttons and other image or pictoral (non-text) toolbox controls, the pszLabel parameter may be used to specify bitmap handles, if the high word of pszLabel is NULL, or bitmap filenames (.bmp, .pcx, or .tif) if the

parameter is a string. Also, the ID parameter may be used to specify pre-defined bitmap types contained in Tool Library; see the DS_ATC_xxx values in toollib.h or toollib.bas, under the comment "built-in button IDs".

The DSAddToolboxControl function adds objects to a toolbox sequentially. The pre-defined DS_ATC_NEWLINE and DS_ATC_SEPARATOR control types can be used to control the spacing and "shape" the toolbox, depending on the toolbox type (box, horizontal bar, vertical bar, etc; see DSCreateToolbox function).

The toolbox size is automatically adjusted as new controls and objects are added. For example, adding one very large control will cause the toolbox to expand, as a rectangle, to accomodate that control in width and height, regardless of the size of other controls.

A call to the DSSetToolboxFlags function, with DS_GTF_TOOLBOXENABLE attribute set, is required to enable a toolbox and make it visible. Without this, no controls or user-objects added to the toolbox will be visible.

The DS_GTF_MENUENABLE attribute will cause a menu item to be added to the toolbox's owner window (unless the owner window is a child window) for each control

or user-object. In this case, the label of the menu item will be the pszLabel parameter.

C/C++ Examples

The examples below show several DSAddToolboxControl function calls being used to add toolbar buttons, edit fields, and other simple controls and user objects to a toolbox. The toolbox handle is assumed to be valid.

```
HWND hdlg;
HTOOLBOX    hPopupBox;
HTOOLBOX    hToolbox;
HCTRLITEMS  hCtrlItems;
PCTRLITEMS  pCtrlItems;

SendMessage(hdlg, WM_SETICON, FALSE,
  (LPARAM)LoadIcon((HINSTANCE)GetWindowLong(hdlg, GWL_HINSTANCE),
  "Smiley"));

// create a toolbar to use for some basic functions
hToolbox = DSCreateToolbox(hdlg, "Controls",
  DS_CT_HBAR | DS_CT_STRIPSTYLE, 0, 0);

// Create a simple toolbar button
DSAddToolboxControl(hToolbox, DS_ATC_VERTSCALEUP,
   DS_ATC_MOMENTARYBUTTON, NULL, NULL, NULL);
DSAddToolboxControl(hToolbox, NULL, DS_ATC_SEPARATOR, NULL,
  NULL, NULL);

// Create a popup box
hPopupBox = DSAddToolboxControl(hToolbox, DS_ATC_FSENTRY,
   DS_ATC_MOMENTARYBUTTON | DS_ATC_POPUP | DS_ATC_3DSTYLE,
   NULL, NULL, NULL);

// create and initialize a CTRLITEMS structure for use with popup control
hCtrlItems = GlobalAlloc(GMEM_ZEROINIT | GMEM_MOVEABLE,
  sizeof(CTRLITEMS));
pCtrlItems = (PCTRLITEMS)GlobalLock(hCtrlItems);

// create a label
DSAddToolboxControl(hPopupBox, NULL,
  DS_ATC_LABEL |DS_ATC_CENTERLABEL, "Num of channels", NULL,
   pCtrlItems);
```

```
// create a text box
lstrcpy(pCtrlItems->szCurEntry, "1");
DSAddToolboxControl(hPopupBox, NULL,
  DS_ATC_TEXTLEFTJUST | DS_ATC_EDITFIELD, NULL, NULL , pCtrlItems );

DSAddToolboxControl(hToolbox, NULL, DS_ATC_SEPARATOR, NULL, NULL,
  NULL);

DSSetToolboxFlags(hToolbox,
  DS_GTF_TOOLBOXENABLE | DS_GTF_MENUENABLE);

GlobalUnlock(hCtrlItems);
GlobalFree(hCtrlItems);
```

See Also

DSCreateToolbox

DSSendMessageControl

DSSendMessageToolboxControl

DSSetToolboxFlags

DSCreateToolbox

HTOOLBOX DSCreateToolbox (hWndOwner,
szTitle
dwType
x
y)

HWND hWndOwner;	// handle of owner window
LPCSTR szTitle;	// pointer to text string containing title of the toolbox
DWORD dwType;	// type of toolbox
short int x;	// x starting positon
short int y;	// y starting position

The DSCreateToolBox function create a new toolbox. The toolbox may be thought of as a "container", which automatically sizes and

shapes depending on the controls added. Objects and controls include toolbar buttons, pop-up dialogs, edit fields, knobs, dials, meters, list boxes, etc.

See below for DSCreateToolbox calling parameter descriptions.

Parameter	Description
hWndOwner	handle of owner window for the toolbox; thw owner window can be a main (overlapped or popup window) or child window
szTitle	text string for toolbox title; a NULL value specifies no toolbox title
dwType	please refer to toollib.h
x	starting x position of toolbox, in owner window coordinates. Additional pre-defined values are shown below:

Value	Meaning
DS_CT_PREVBOX	use previous box for horizontal start position
DS_CT_PREVBOXADJACENT	use previous box for horizontal start position, adjacent
DS_CT_RIGHT	align toolbox with right of owner window

y — starting y position of toolbox, in owner window coordinates. Additional pre-defined values are shown below:

Value	Meaning
DS_CT_PREVBOX	use previous box for vertical start position
DS_CT_PREVBOXADJACENT	use previous box for vertical start position, adjacent
DS_CT_BOTTOM	align toolbox with bottom of owner window

Returns

The return value is handle to the toolbox created, if successful. Otherwise, the return value is NULL, indicating the owner handle is invalid.

For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSCreateToolBox function to create a toolbox:

```
HWND        hdlg;
HTOOLBOX hToolbox;

hToolbox = DSCreateToolbox(hdlg, "Controls",
  DS_CT_HBAR | DS_CT_STRIPSTYLE, 0, 0);
```

Note that

See Also

DSAddToolboxControl

DSDeleteToolbox

DSGetToolboxFlags

DSSetToolboxFlags

DSDeleteToolbox

UINT DSDeleteToolbox (hToolbox)

HTOOLBOX hToolbox // handle of toolbox

The DSDeleteToolbox function detete the toolbox with the handle hToolbox.

Parameter	Description
hToolbox	handle of toolbox to be destroyed

Returns

The return value is handle to the toolbox created, if successful. Otherwise, the return value is NULL, indicating the owner handle is invalid.

For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSDeleteToolbox function:

```
HWND        hdlg;
HTOOLBOX hToolbox;

hToolbox = DSCreateToolbox(hdlg, "Controls",
  DS_CT_HBAR | DS_CT_STRIPSTYLE, 0, 0);
if (hToolbox != NULL) DSDeleteToolbox(hToolbox);
```

Note that

See Also

DSCreateToolbox

DSGetToolboxFlags

UINT DSGetToolboxFlags (hToolbox)

HTOOLBOX hToolbox; // toolbox handle

The DSGetToolboxFlags function retrieves the flags of the toolbox.

Parameter	Description
Htoolbox	handle of toolbox

Returns

The return value is flags of the toolbox, if successful. Otherwise, the return value is NULL, indicating toolbox handle is invalid.

Value	Meaning
DS_GTF_TOOLBOXENABLE	// enable toolbox (show it)
DS_GTF_MENUENABLE	// enable toolbox's menu
DS_GTF_MOVEABLE	// make toolbox moveable/fixed
DS_GTF_TOPPARENTREL	// position and clip toolbox relative to top parent
DS_GTF_NONCLIENTAREA	// toolbox must be popup, because it will be located in non-client area
DS_GTF_MOVEWITHOWNER	// move if owner control is moved using DSMoveToolbox
DS_GTF_BORDER	// border attribute
DS_GTF_BORDERSPECIAL	// special red oval border used by DSPower-Block Diagram
DS_GTF_NOTVISIBLE	// toolbox should not be visible even after being enabled (becomes calling app's responsibility to control visibility)
DS_GTF_3DSTYLE	// activate 3D style (border, background)
DS_GTF_NOCLOSEBUTTON	// can be used to override Close button attribute
DS_GTF_NOOKBUTTON	// can be used to override OK button attribute

DS_GTF_SHADEDBKGND	// toolbox background is shaded (see DS_CT_SHADEDBKGND)
DS_GTF_NOAUTOSTICK	// disable "autostick", which occurs when toolbox shares two borders with owner window (toollib keeps it there, if owner window changes size)

For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSGetToolboxFlags function:

```
HWND hdlg;
HTOOLBOX hToolbox;
UINT uFlags;

hToolbox = DSCreateToolbox(hdlg, "Controls",
  DS_CT_HBAR | DS_CT_STRIPSTYLE, 0, 0);
uFlags = DSGetToolboxFlags(hToolbox);
```

Note

See Also

DSAddToolboxControl

DSCreateToolbox

DSSetToolboxFlags

DSGetToolboxRect

UINT DSGetToolboxRect (hToolbox, prect)

HTOOLBOX hToolbox;	// toolbox handle

RECT far* prect; // pointer to rectangle coordinates

The DSGetToolboxRect function retrieve the client coordinate of the toolbox

Parameter	Description
Htoolbox	handle of toolbox
Prect	pointer to client rectangle coordinates

Returns

The return value is TRUE if successful. Otherwise, the return value is FALSE, indicating the toolbox handle is invalid.

For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSGetToolboxRect function to retrieve the client coordinate of toolbox rectangle:

```
HWND hdlg;
HTOOLBOX hToolbox;
RECT rect;

hToolbox = DSCreateToolbox(hdlg, "Controls",
  DS_CT_HBAR | DS_CT_STRIPSTYLE, 0, 0);

// return the client coordinate of toolbox
```

DSGetToolboxRect(hToolbox, &rect);

Note

See Also

DSAddToolboxControl

DSCreateToolbox

DSMoveToolbox

DSGetToolLibErrorStatus

UINT DSGetToolLibErrorStatus ( )

The DSGetToolLibErrorStatus function return the errors associated with running toollib.dll.

Parameter	Description
None	

Returns

The return value is errors contained in dllerr.h.

Value	Meaning
DSCOULDNOTCREATEBKGNDWINDOW	
DSCOULDNOTSTARTBKGNDTASK	
DSNOBKGNDWINDOW	
DSCOULDNOTINITIALIZEDLL	

Comments

The

C/C++ Example

Below is an example of using the DSGetToolLibErrorStatus function:

```
HWND hWnd;
UINT ret;

hWnd = DSInitToolLib();
```

if(hWnd == NULL) ret = DSGetToolLibErrorStatus();

Note that

DSGetToolLibFlags

UINT DSGetToolLibFlags ( )

The DSGetToolLibFlags function retrieves the flags that affects toollib.dll.

Parameter	Description
None	

Returns

The return value is flags associated with a toolbox.

Value	Meaning
DS_GTGF_SIZEINDEPENDENT	// object size independent of screen resolution
DS_GTGF_FLYOVERHELPENABLE	// enable fly-over help
DS_GTGF_FLYOVERHELPMINIMIZE	// enable short-form of fly-over help
DS_GTGF_ENABLECONTEXTHELP	// enable small context-sensitive help button
DS_GTGF_WIN31CONTEXTHELPLOOK	// enable Win 3.1x context-sensitive help look (solid yellow, instead of light yellow)

For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSGetToolLibFlags function:

```
HWND hWnd;
UINT uFlags;

hWnd = DSInitToolLib();
uFlags = DSGetToolLibFlags();
```

Note that

See Also

DSSetToolLibFlags

DSGetUserInput

int LIBAPI DSGetUserInput (hdlg,
szTitle
szLabel
szText
cbLen)

HWND hdlg;	// handle of owner dialog
LPCSTR szTitle;	// title of user input dialog
LPCSTR szLabel;	// label of edit field in user input dialog
LPSTR szText;	// default text in edit field
int cbLen;	// length of edit field

The DSGetUserInput function

initialize a dialog that retrieves information from the user.

Parameter	Description
Hdlg	handle of owner dialog
SzTitle	pointer to null-terminated text string which will contains title of the User Input dialog
SzLabel	pointer to null-terminated text string which will contains label of the Text Edit Field
SzText	pointer to null-terminated text string which will contains default text in the Text Edit field
CbLen	length of Text Edit field

Returns

The return value is the return value is the value of the nResult parameter given in the call to the EndDialog function used to terminate the dialog box, if successful.??? Otherwise, the return value is -1. For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSGetUserInput:

```
HWND hWnd;
char      szLabel[100], tmpstr1[256], tmpstr2[256];

// example code of using DSGetUserInput
```

```
lstrcpy(szLabel, "Please enter your name");
lstrcpy(tmpstr1, "John Smith");

DSGetUserInput(hdlg, "DSGetUserInput Example", szLabel, tmpstr1,
  sizeof(tmpstr1));

lstrcpy(szLabel, "DSGetUserInput");
lstrcpy(tmpstr2, "You name is ");
lstrcat(tmpstr2, tmpstr1);

DSMessageBox(NULL, tmpstr2, szLabel, MB_SYSTEMMODAL);
```

Note that

See Also

DSMessageBox

DSInitToolLib

HWND DSInitToolLib ( )

The DSInitToolLib function invokes toollib.dll and runs it in the background.

Parameter	Description
None	

Returns

The return value is handle to the toollib, if successful. Otherwise, the return value is NULL, indicating background handle is invalid. For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSInitToolLib:

```
DSInitToolLib();
DSSetToolLibFlags(DS_GTGF_FLYOVERHELPMINIMIZE |
  DS_GTGF_FLYOVERHELPENABLE);  // enable fly-over help
```

Note that

See Also

DSGetToolLibFlags

DSSetToolLibFlags

DSMarkApp

void DSMarkApp (hInst,
wVer)

HINSTANCE hInst;
WORD wVer;

The DSMarkApp function performs run time marking so that 16-bit version of software can use all the new functionality of their 32-bit counterpart.

Parameter	Description
Hinst	
WVer	

Returns

The return value is handle to the toolbox created if successful. Otherwise, the return value is NULL, indicating the Hardware Setup File index was out of range. For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSMarkApp function:

```
BOOL fWin32;
HANDLE hInstance;

// set Win 3.1x / Win95 flag
fWin32 = ((LOWORD(GetVersion() & 0xff) == 3) &&
  ((LOWORD(GetVersion()) >> 8) >= 95));

// if Win95, mark app as 4.0
if (fWin32) DSMarkApp(hInstance, 0x0400);
```

Note that

See Also

DSGetToolLibFlags

DSInitToolLib

DSSetToolLibFlags

DSMessageBox

int DSMessageBox (hwndParent,
lpszText
lpszTitle

dwStyle)

```
HWND hwndParent;    // parent windle handle
LPCSTR lpszText;    // pointer to text string
LPCSTR lpszTitle;   // pointer to title string
DWORD dwStyle;      // valid dialog style
```

The DSMessageBox function

Parameter	Description
hwndParent	Identifies the owner window of the message box to be created. If this parameter is NULL, the message box has no owner window.
lpszText	Points to a null-terminated string containing the message to be displayed.
lpszTitle	Points to a null-terminated string used for the dialog box title. If this parameter is NULL, the default title Error is used.
dwStyle	Specifies the contents and behavior of the dialog box. This parameter can be a combination of the following values:

Value	Meaning
MB_ABORTRETRYIGNORE	Message box contains Abort, Retry, and Ignore buttons
MB_APPLMODAL	User must respond to the message box before continuing work in the window identified by the hwndParent parameter
MB_APPLMODAL	Default value if neither MB_SYSTEMMODAL nor MB_TASKMODAL is specified
MB_DEFAULT_DESKTOP_ONLY	The desktop currently receiving input must be a default desktop; otherwise, the function fails. A default desktop is one an application runs on after the user has logged on
MB_DEFBUTTON1	First button is the default button. Note that the first button is always the default unless MB_DEFBUTTON2, MB_DEFBUTTON3, or MB_DEFBUTTON4 is specified
MB_DEFBUTTON2	Second button is a default button

MB_DEFBUTTON3	Third button is a default button
MB_DEFBUTTON4	The fourth button is a default button
MB_HELP	Windows 95 only: Adds a Help button to the message box. Choosing the Help button or pressing F1 generates a Help event
MB_ICONASTERISK	Same as MB_ICONINFORMATION
MB_ICONERROR	Windows 95 only: Same as MB_ICONHAND
MB_ICONEXCLAMATION	An exclamation-point icon appears in the message box
MB_ICONHAND	Same as MB_ICONSTOP
MB_ICONINFORMATION	An icon consisting of a lowercase letter i in a circle appears in the message box
MB_ICONQUESTION	A question-mark icon appears in the message box
MB_ICONSTOP	A stop-sign icon appears in the message box
MB_ICONWARNING	Windows 95 only: Same as MB_ICONEXCLAMATION
MB_OK	The message box contains one push button: OK
MB_OKCANCEL	The message box contains two push buttons: OK and Cancel.
MB_RETRYCANCEL	Message box contains two push buttons: Retry and Cancel.
MB_RIGHT	Windows 95 only : The text is right-justified
MB_RTLREADING	Windows 95 only : Displays message and caption text using right-to-left reading order on Hebrew and Arabic systems.
MB_SERVICE_NOTIFICATION	Windows NT only : The caller is a service notifying the user of an event. The function displays a message box on the current active desktop, even if there is no user logged on to the computer.If this flag is set, the hWnd parameter must be NULL. This is so the message box can appear on a desktop other than the desktop corresponding to the hWnd.
MB_SETFOREGROUND	Message box becomes the foreground window. Internally, Windows calls the SetForegroundWindow function for the message box.

MB_SYSTEMMODAL	All applications are suspended until the user responds to the message box. Unless the application specifies MB_ICONHAND, the message box does not become modal until after it is created; consequently, the owner window and other windows continue to receive messages resulting from its activation. Use system-modal message boxes to notify the user of serious, potentially damaging errors that require immediate attention (for example, running out of memory).
MB_TASKMODAL	Same as MB_APPLMODAL except that all the top-level windows belonging to the current task are disabled if the hWnd parameter is NULL. Use this flag when the calling application or library does not have a window handle available but still needs to prevent input to other windows in the current application without suspending other applications.
MB_YESNO	The message box contains two push buttons: Yes and No.
MB_YESNOCANCEL	The message box contains three push buttons: Yes, No, and Cancel.

Returns

The return value is zero if there is not enough memory to create the message box. If the function succeeds, the return value is one of the following menu-item values returned by the dialog box:

Value	Meaning
IDABORT	Abort button was selected
IDCANCEL	Cancel button was selected
IDIGNORE	Ignore button was selected
IDNO	No button was selected
IDOK	OK button was selected
IDRETRY	Retry button was selected
IDYES	Yes button was selected

If a message box has a Cancel button, the function returns the IDCANCEL value if either the ESC key is pressed or the Cancel button is selected. If the message box has no Cancel button, pressing ESC has no effect.

The return value is handle to the toolbox created if successful. Otherwise, the return value is NULL, indicating the Hardware Setup File index was out of range. For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using DSMessageBox function

```
DSMessageBox(NULL, “Text in Message Box”, “Title text”,
  MB_OK | MB_ALWAYSONTOP|MB_SYSTEMMODAL);
```

Note that

See Also

DSGetUserInput

DSMoveToolbox

UINT DSMoveToolbox (hToolbox,
x

```
                                    y
                                    wd
                                    ht )

HTOOLBOX hToolbox;          // handle of window
short int x;                // horizontal position
short int y;                // vertical position
short int wd;               // width
short int ht;               // height
```

The DSMoveToolbox function

Parameter	Description
hToolbox	Identifies the window.
x	Specifies the new position of the left side of the window.
y	Specifies the new position of the top of the window.
wd	Specifies the new width of the window, in pixels.
ht	Specifies the new height of the window, in pixels.

Returns

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. if toolbox handle is invalid.

The SetWindowPos function changes the size, position, and Z order of a child, pop-up, or top-level window. Child, pop-up, and top-level windows are ordered according to their appearance on the screen. The topmost window

receives the highest rank and is the first window in the Z order.

SWP_NOACTIVATE Does not activate the window. If this flag is not set, the window is activated and moved to the top of either the topmost or non-topmost group (depending on the setting of the hwndInsertAfter parameter).

SWP_NOZORDER Retains the current Z order (ignores the hwndInsertAfter parameter).

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To get extended error information, call GetLastError.

Remarks

If the SWP_SHOWWINDOW or SWP_HIDEWINDOW flag is set, the window cannot be moved or sized.

All coordinates for child windows are client coordinates (relative to the upper-left corner of the parent window's client area).

A window can be made a topmost window either by setting the hwndInsertAfter parameter to HWND_TOPMOST and ensuring

that the SWP_NOZORDER flag is not set, or by setting a window's position in the Z order so that it is above any existing topmost windows. When a non-topmost window is made topmost, its owned windows are also made topmost. Its owners, however, are not changed.

If neither the SWP_NOACTIVATE nor SWP_NOZORDER flag is specified (that is, when the application requests that a window be simultaneously activated and its position in the Z order changed), the value specified in hwndInsertAfter is used only in the following circumstances:

· Neither the HWND_TOPMOST nor HWND_NOTOPMOST flag is specified in hwndInsertAfter.

· The window identified by hwnd is not the active window.

An application cannot activate an inactive window without also bringing it to the top of the Z order. Applications can change an activated window's position in the Z order without restrictions, or it can activate a window and then move it to the top of the topmost or non-topmost windows.

If a topmost window is repositioned to the bottom (HWND_BOTTOM) of the Z order or after any non-

topmost window, it is no longer topmost. When a topmost window is made non-topmost, its owners and its owned windows are also made non-topmost windows.

A non-topmost window can own a topmost window, but the reverse cannot occur. Any window (for example, a dialog box) owned by a topmost window is itself made a topmost window, to ensure that all owned windows stay above their owner.

If an application is not in the foreground, and should be in the foreground, it must call the SetForegroundWindow function.

The return value is handle to the toolbox created if successful. Otherwise, the return value is NULL, indicating the Hardware Setup File index was out of range. For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSMoveToolbox:

```
HWND hdlg;
HTOOLBOX hToolbox;
```

```
hToolbox = DSCreateToolbox(hdlg, "Controls",
  DS_CT_HBAR | DS_CT_STRIPSTYLE, 0, 0);

// move toolbox left to 5, and top to 10 and retain original height and width
DSMoveToolbox(hToolbox, 5, 10, NULL, NULL);

// move toolbox left to 5, and top to 10 and change width to 100, height to 50
DSMoveToolbox(hToolbox, 5, 10, 100, 50);
```

Note that

See Also

DSAddToolboxControl

DSCreateToolbox

DSGetToolboxRect

DSSendMessageControl

LRESULT DSSendMessageControl (hControl,
uMsg
wParam
lParam)

HCONTROL hControl;
UINT uMsg;
WPARAM wParam;
LPARAM lParam; //

The DSSendMessageControl function

Parameter	Description
hControl	
uMsg	
wParam	

lParam

Returns

The return value is handle to the toolbox created if successful. Otherwise, the return value is NULL, indicating the Hardware Setup File index was out of range. For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSSendMessageControl:

```
HWND hdlg;
HTOOLBOX hToolbox;
HTOOLBOX    hCheck1, hCheck2;

hToolbox = DSCreateToolbox(hdlg, "Controls",
  DS_CT_HBAR | DS_CT_STRIPSTYLE, 0, 0);

// Create simple checkboxes
hCheck1 = DSAddToolboxControl(hToolbox, NULL, DS_ATC_CHECKBOX ,
  "Unchecked" , "Check Box 1", NULL);
hCheck2 = DSAddToolboxControl(hToolbox, NULL, DS_ATC_CHECKBOX ,
  "Checked", "Check Box 2", NULL);

// Set check box state to unchecked
DSSendMessageControl(hCheck1, BM_SETCHECK, (WPARAM)0, 0L);
// Set check box state to checked
DSSendMessageControl(hCheck2, BM_SETCHECK, (WPARAM)1, 0L);
```

Note that

See Also

DSAddToolboxControl

DSSendMessageToolboxControl

DSSendMessageToolbox Control

LRESULT DSSendMessageToolboxControl (hToolbox,
ID
uMsg
wParam
lParam)

HTOOLBOX hToolbox;
WORD ID;
UINT uMsg;
WPARAM wParam;
LPARAM lParam; //

The DSSendMessageToolboxControl function

Parameter	Description
HToolbox	
ID	
UMsg	
WParam	
LParam	

Returns

The return value is handle to the toolbox created if successful. Otherwise, the return value is NULL, indicating the???. For

further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSSendMessageToolboxControl:

```
HWND hdlg;
HTOOLBOX hToolbox;
HCTRLITEMS  hCtrlItems;
PCTRLITEMS  pCtrlItems;

hToolbox = DSCreateToolbox(hdlg, "Controls",
  DS_CT_HBAR | DS_CT_STRIPSTYLE, 0, 0);

// create and initialize a CTRLITEMS structure for use with popup control
hCtrlItems = GlobalAlloc(GMEM_ZEROINIT | GMEM_MOVEABLE,
  sizeof(CTRLITEMS));
pCtrlItems = (PCTRLITEMS)GlobalLock(hCtrlItems);

// create a text box
// Set initial text to "11111111111"
lstrcpy(pCtrlItems->szCurEntry, "11111111111");
hText = DSAddToolboxControl(hToolbox, NULL,
  DS_ATC_TEXTLEFTJUST | DS_ATC_EDITFIELD, NULL, NULL , pCtrlItems );
DSSetToolboxFlags(hToolbox,
  DS_GTF_TOOLBOXENABLE | DS_GTF_MENUENABLE);

lstrcpy(pCtrlItems->szCurEntry, "Hello");  // Change control text to "Hello"
DSSendMessageToolboxControl(hToolbox, 100, WM_SETTEXT, 0L,
  (LPARAM)pCtrlItems->szCurEntry);

GlobalUnlock(hCtrlItems);
GlobalFree(hCtrlItems);
```

Note that

See Also

DSAddToolboxControl

DSSetToolboxFlags

UINT DSSetToolboxFlags (hToolbox,
Flags)

HTOOLBOX hToolbox;
UINT Flags;

The DSSetToolboxFlags function

Parameter	Description
Htoolbox	
Flags	

Returns

The return value is handle to the toolbox created if successful. Otherwise, the return value is NULL, indicating ?????

For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSSetToolboxFlags:

```
HWND hdlg;
```

```
HTOOLBOX hToolbox;

hToolbar = DSCreateToolbox(hdlg, "Controls",
  DS_CT_HBAR | DS_CT_STRIPSTYLE, 0, 0);
DSSetToolboxFlags(hToolbox,
  DS_GTF_TOOLBOXENABLE | DS_GTF_MENUENABLE);
```

Note that

See Also

DSCreateToolbox

DSGetToolboxFlags

DSSetToolLibFlags

void DSSetToolLibFlags (Flags)

UINT Flags;

The DSSetToolLibFlags function

Parameter	Description
Flags	#define DS_GTGF_SIZEINDEPENDENT 1 // object size independent of screen resolution #define DS_GTGF_FLYOVERHELPENABLE 2 // enable fly-over help #define DS_GTGF_FLYOVERHELPMINIMIZE 4 // enable short-form of fly-over help #define DS_GTGF_ENABLECONTEXTHELP 8 // enable small context-sensitive help button #define DS_GTGF_WIN31CONTEXTHELPLOOK 16 // enable Win 3.1x context-sensitive help look (solid yellow, instead of light yellow)

Returns

The return value is handle to the toolbox created if successful.

Otherwise, the return value is NULL, indicating ?????. For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSSetToolLibFlags:

```
DSInitToolLib();
DSSetToolLibFlags(DS_GTGF_FLYOVERHELPMINIMIZE |
  DS_GTGF_ENABLECONTEXTHELP);
```

Note that

See Also

DSGetToolLibFlags

DSInitToolLib

2.2.5 Signal Processing Library Functions

@k functions,, DSPower-SPLib, functions, Signal Processing Library, splib.dll]

This section not available yet.

2.3 DSP Source Code Functions

This section not yet available. For general information, please see Signalogic DSP Software Users Guide chapter 5, Using DSP Source Code Interface Software, which contains the general and specific topics sections of interest listed below:

General DSP Source Code Interface Software TopicsSections

5.0.1 DSP Source Code Interface Documentation

5.0.2 DSP Source Code Interfaces for Hypersignal

5.0.3 DSP Source Code Interfaces for Hypersignal in Engine Mode

Specific DSP Source Code Interface Software TopicsSections

5.7 Analog Devices ADSP-21xx Source Code Interface for Hypersignal

5.2 AT&T WE DSP32 Source Code Interface for Hypersignal

5.4 AT&T WE DSP32C Source Code Interface for Hypersignal

5.9 AT&T WE DSP3210 Source Code Interface for Hypersignal

5.3 Motorola DSP5600x Source Code Interface for Hypersignal

5.8 Motorola DSP96002 Source Code Interface for Hypersignal

5.6 NEC 77220 Source Code Interface for Hypersignal

5.1 Texas Instruments TMS320C25 Source Code Interface for Hypersignal

5.5 Texas Instruments TMS320C3x Source Code Interface for Hypersignal

5.10 Texas Instruments TMS320C4x Source Code Interface for Hypersignal

5.11 Texas Instruments TMS320C5x Source Code Interface for Hypersignal

3 DSP/Analog Hardware Reference

Overview

[@j DSP/Analog Hardware Reference Overview]

DSP/Analog Hardware Reference is a collection of information which may be helpful when creating user-defined C/C++, MATLAB, or Visual Basic applications need to work with supported DSP/analog hardware.

This chapter section is *programming-oriented, and not operation-oriented.* It is not intended to provide information about individual hardware types. For detailed information about specific DSP/analog hardware types, see section 3.1 below, DSP/Analog Hardware Documentation.

Main Topics

DSP/Analog Hardware Documentation
Board Designator, Driver ID, and Classification Table
Hypersignal DSP Source Code Variable List

3.1 DSP/Analog Hardware Documentation

As mentioned in the DSP/Analog Hardware Reference Overview section above, information in this chapter section is not intended to speak to individual DSP/analog hardware types. Here is a list of places to look for additional DSP/analog hardware information:

Hypersignal-Macro Series Hardware Reference Guide (most complete source by far)

Signalogic DSP Software Users Guide chapter 4, Using DSP Source Code Interface Software

section 3.2 below, Board Designator, Driver ID, and Classification Table

section 3.3 below, Hypersignal DSP Source Code Variable List

Signalogic DSP Software System Guide section 4.3, Troubleshooting DSP/Analog Hardware Problems

Signalogic DSP Software System Guide section 4.3.1, I/O-Mapped Hardware Conflicts

Signalogic DSP Software System Guide section 4.3.2, Memory-Mapped Transfer Gotchas

Signalogic DSP Software System Guide section 4.3.3, Host PC Internal Bus Speed Conflicts

Signalogic DSP Software System Guide section 4.3.4, Data Acquisition Problems

Signalogic DSP Software Users Guide section 4.3, DSPower-HWLib Function Overview

Signalogic DSP Software Users Guide section 4.5, Acquiring Data Using DSPower-HWLib Software

Signalogic DSP Software Users Guide section 8.2, Hardware Selector

Signalogic DSP Software Users Guide section 8.3, Hardware Setup File

DSP/Analog Hardware Solutions Guide (marketing brochure which contains partial summary of DSP/analog hardware types)

3.2 Board Designator, Driver ID, and Classification Table

Listed below are current valid Board Designators which can be used in the DSAssignBoard, DSGetHWMgrCurVal, DSSetHWMgrCurVal, DSShowHardwareSelectorDlg, and DSGetHWMgrEntryIndex calls in Hardware Library and Hardware Manager (DSPower-HWLib software).

Values corresponding to Board Designators which are also shown include description, manufacturer, Driver IDs, DSP code classification and subclassification, and default DSP Program File. The table below is organized alphabetically by manufacturer (as is the Hardware Selector in Hardware Manager), to hopefully make finding Board Designators easier.

Manufacturer	Board Designator	Description	Driver IDs	Cls	Default DSP Program File
Ariel	ARI320	DSP-16	2,0	1,0	tms32025.out
Ariel	ARI32C	DSP-32C	30,0	1,0	dsp32c.out
Ariel	ARI56	DSP-56	29,4	5,0	dsp56000.out
Ariel	ARI96	DSP-96	80,0	1,0	dsp96000.out
Ariel	ARI3210	MP3210	85,0	0,0	dsp3210.out
Ariel	ARI3210/1	MP3210 (second	85,0	0,0	dsp3210.out

		processor)			
Ariel	ARIC40	DSP-C40	45,0	0,0	tms32040.out
Ariel	ARIC40-B	DSP-C40U (SRAM only)	45,4	0,0	tms32040.out
Ariel	ARI320-B	PC-C25	2,4	1,1	tms32025.out
Ariel	ARI56-B	PC-56	29,0	2,0	dsp56000.out
Ariel	ARI56-B	PC-56D with onboard I/O	29,0	2,0	dsp56000.out
Ariel	ARI56-D	DatPort with PC-56D	29,6	6,0	dsp56000.out
Ariel	ARI56-D	ProPort with PC-56D	29,6	6,0	dsp56000.out
Ariel	ARI56-DX	ProPort w/ PC-56D, 100 kHz	29,6	6,0	dsp56000.out
Ariel	ARI3210-B	ProPort w/ MP3210	85,0	0,0	dsp3210.out
Ariel	ARI56-DM	Dig. Microphone with PC-56D	29,38	6,0	dsp56000.out
Ariel	ARI32C-B	ProPort with PC-32C	30,4	1,1	dsp32c.out
Ariel	ARI32C-B	DatPort with PC-32C	30,4	1,1	dsp32c.out
AT&T	ATT32C	DSP32C development board	31,0	2,0	dsp32c.out
ASPI	ASPIC30	Banshee (TMS-320C30)	59,0	1,0	tms32030.out
ASPI	ASPIC30-AD16	Banshee with AD16 daughterbd	59,0	1,0	tms32030.out
CAC	CAC32	DSP32-PC	10,0	0,0	dsp32.out
CAC	CAC32C	DSP32C-AT	32,0	3,0	dsp32c.bin
CAC	CAC32C-COD	AC5 with CODEC daughterbd	32,6	3,3	dsp32c.out
CAC	CAC32C-DBD	AC5 with DBDADA daughterbd	32,5	3,2	dsp32c.out
CAC	CAC32C-DBC	AC5 with DBCS5339 daughterbd	32,4	3,1	dsp32c.out
CAC	CAC3210	QUANTUMdsp DSP3210	86,0	1,0	dsp3210.out
Data Translation	DT2821	DT2821 series	70,0	N/A	N/A
Data Translation	DT2823	DT2823 series	70,0	N/A	N/A
Digital Audio Labs	DRB1	Digital	72,0	N/A	N/A

		Recording Board I			
Digital Audio Labs	DRB2	Digital Recording Board II	72,0	N/A	N/A
DSP Research	DRIC30	TIGER 30	53,0	4,0	tms32030.out
DSP Research	DRIC30-BB71	TIGER 30 with BB71 (rev. B)	53,5	4,2	tms32030.out
Domain Technologies	DOMC31	DSPCard-C31	56,0	3,0	tms32030.out
Electronic Tools	ET56	miniKit 56001	23,0	9,0	dsp56000.out
Innovative Integration	IIC31	PC31	55,0	6,0	tms32030.out
Innovative Integration	IIC31-B	PC31, no dual-port SRAM	55,4	6,0	tms32030.out
Innovative Integration	IIC50	PC50	5,0	0,0	tms32050.out
Innovative Integration	IIC50-B	PC50, no dual-port SRAM	5,4	0,0	tms32050.out
Intelligent Instrumentation	PCI41C	PCI20041C (auto config)	60,0	N/A	N/A
Intelligent Instrumentation	PCI202C	PCI20202C (auto config)	9,0	3,0	tms32025.out
Intelligent Instrumentation	ZPB32-ZPB100	ZPB32 (DSP32) with ZPB100	14,0	1,0	dsp32.bin
Intelligent Instrumentation	ZPB32-ZPD1002	ZPB32 (DSP32) with ZPD1002	14,0	1,0	dsp32.bin
Intelligent Instrumentation	ZPB32-ZPD1007	ZPB32 (DSP32) with ZPD1007	14,0	1,0	dsp32.bin
Intelligent Instrumentation	ZPB32-ZPB1100	ZPB32 (DSP32) with ZPB1100	14,4	1,0	dsp32.bin
Intelligent Instrumentation	ZPB32-ZPB2100	ZPB32 (DSP32) with ZPB2100	14,4	1,0	dsp32.bin
Intelligent Instrumentation	ZPB34-ZPB100	ZPB34A (DSP32C) with ZPB100	33,6	4,3	dsp32c.out
Intelligent Instrumentation	ZPB34-ZPD1002	ZPB34A (DSP32C) with ZPD1002	33,0	4,0	dsp32c.out
Intelligent Instrumentation	ZPB34-ZPD1007	ZPB34A (DSP32C) with ZPD1007	33,3	4,0	dsp32c.out
Intelligent Instrumentation	ZPB34-ZPB1100	ZPB34A (DSP32C) with ZPD1100	33,4	4,2	dsp32c.out
Intelligent Instrumentation	ZPB34-ZPB2100	ZPB34A (DSP32C) with ZPD2100	33,4	4,2	dsp32c.out

Intelligent Instrumentation	ZPB34-ZPB6064	ZPB34A (DSP32C) with ZPB6064	33,5	4,1	dsp32c.out
Loughborough Sound Images	LSI320	TMS320C25 system board	0,0	0,0	tms32025.out
Loughborough Sound Images	LSI56	DSP56001 system board	20,0	0,0	dsp56000.out
Loughborough Sound Images	LSI32C	DSP32C system board	39,0	0,0	dsp32c.out
Loughborough Sound Images	LSIC30	TMS320C30 system board	50,0	0,0	tms32030.out
Loughborough Sound Images	LSI2101	ADSP-2101 system board	44,0	0,0	adsp21x.exe
Loughborough Sound Images	LSI96	DSP96002 system board	84,0	0,0	dsp96000.out
Motorola	ADS56-A	ADS dev sys with Ariel ADC56000	21,0	1,0	dsp56000.out
Motorola	ADS56-B	ADS dev sys with ADC16 eval I/O	21,4	4,0	dsp56000.out
MTT Instrumentation	MTT32C	Lory DSP4100	34,0	5,0	dsp32c.out
MTT Instrumentation	MTTC30	Lory DSP4200	54,0	5,0	tms32030.out
National Instruments	NAT32C	AT-DSP2200	35,0	6,0	dsp32c.out
Peralex	PA56	PROTO-56	29,5	2,0	dsp56000.out
Quantawave	QWV32C	Qw32C-8AFA	36,0	7,0	dsp32c.out
Signalogic	SIG32C	Sig32C-8	36,0	7,0	dsp32c.out
Sonitech	SONC30	Spirit-30	51,0	2,0	tms32030.out
Sonitech	SONC30-SAIB	Spirit-30 with SAIB	51,4	2,1	tms32030.out
Sonitech	SONC30-STIB	Spirit-30 with STIB	51,4	2,1	tms32030.out
Techron	TEF20HI	TEF20HI with direct mic input	21,6	8,0	dsp56000.out
Texas Instruments	SWDS	SWDS (TMS-320C25)	1,0	2,0	tms32025.out
Texas Instruments	SWDS-AIB	SWDS with AIB-2 input board	1,0	2,0	tms32025.out
Texas Instruments	EVMC30	EVM (TMS-320C30)	52,0	3,0	tms32030.out
Turtle Beach Systems	TBS56	MultiSound	22,0	3,0	dsp56tbs.out
Turtle Beach	TBS56-B	Tahiti	22,4	3,0	dsp56tbs.out

Notes

1) **Cls** in the table heading above is an abbreviation for **classification**. Classification values above are shown as "main,sub" values. The "main" value can be used with the DSP_BOARDCLASS DSP source code variable, and the "sub" value with the DSP_BOARDSUBCLASS variable. Note that the DSGetBoardClass call in Hardware Library returns a WORD value as follows:

DSGetBoardClass =>> main∗256 + sub

2) N/A (not applicable) as used above means the hardware type does not have a DSP device.

3) Driver ID values above are shown "major,minor" values. Note that the DSGetBoardInfo call with DS_GBI_DRIVERID parameter returns a Driver ID value in the form:

wDriverID = minor∗256 + major

For more information see Driver ID Fields in Hardware Setup File, in Signalogic DSP Software Users Guide section 8.3.1, Hardware Setup File Fields.

4) Note that in some cases similar DSP/analog hardware types look identical to host computer (Driver ID values are same), but different from the onboard DSP device's point of view. This typically occurs when the same manufacturer builds boards with differ only with analog

I/O capabilities. On the other hand, some DSP/analog hardware types look the same from to the onboard DSP, but different from the host's point of view. This usually occurs when different manufacturers build boards which have similar DSP, memory, and analog I/O characteristics.

3.3 Hypersignal DSP Source Code Variable List

Listed below are pre-defined constants which can be used in DSGetHVarMem and DSPutHVarMem calls in Hardware Library (DSPower-HWLib software), in order to control and communicate with DSP code executing in conjunction with Hypersignal acquisition, generation, instrument, and file-based acceleration functions. These calls, as well as the DSGetMem and DSPutMem calls, can be invoked either while the DSP is executing or in a static state.

Related Documentation TopicsSections

Signalogic DSP Software Users Guide section 5.0.2, DSP Source Code Interfaces for Hypersignal

Signalogic DSP Software Users Guide section 5.0.3, DSP Source Code Interfaces for Hypersignal in Engine Mode

Hypersignal-Macro Series Hardware Reference Guide chapter 4, DSP Source Code Interface

Hypersignal-Macro Series Hardware Reference Guide Appendix A, WriteDSPVal Constants

See Also

DSGetHVarMem

DSPutHVarMem

Hypersignal DSP Source Code Variables

DSP_ADCOUNT

DSP_BOARDCLASS

DSP_BOARDSUBCLASS

DSP_BUFLEN

DSP_BUFNUM

DSP_BUSYFLG

DSP_CHANLIST

DSP_CODECCTRL1

DSP_CODECCTRL2

DSP_COUPLIST

DSP_DACOUNT

DSP_DUPFLG

DSP_FILTADDR1

DSP_FILTADDR2

DSP_FILTLEN1

DSP_FILTLEN2

DSP_FILTTYPE1

DSP_FILTTYPE2

DSP_FILTQUANT

DSP_FILTUPDFLG

DSP_FFTLEN

DSP_FFTORD

DSP_FRMSIZ

DSP_FSMODE

DSP_FSVALUE
DSP_GAINLIST
DSP_HOSTBUFNUM
DSP_IOMODULE
DSP_LOGCOEFFA
DSP_LOGCOEFFB
DSP_LOGFLG1
DSP_LOGFLG2
DSP_MAGDATAADDR
DSP_MAXVAL
DSP_MAXVAL1
DSP_MAXVAL2
DSP_MAXVALIMAG
DSP_MAXVALREAL
DSP_MINVAL
DSP_MONITORFLG
DSP_NUMCHAN
DSP_OFFSETIN
DSP_OFFSETOUT
DSP_OPMODE
DSP_OVERFLOW
DSP_PHZREQ
DSP_PWRCOEFFA
DSP_PWRCOEFFB
DSP_PWRFLG1
DSP_PWRFLG2
DSP_RIDATAADDR
DSP_RIFLG
DSP_SCALEIN

DSP_SCALEOUT

DSP_SERIALCTRL

DSP_STMBUFLEN

DSP_STMDATAADDR

DSP_TIMDATAADDR

DSP_TRIGCHANLIST

DSP_TRIGLEVEL

DSP_WINSCL

DSP_WINDATAADDR

DSP_XFERFLG1

DSP_XFERFLG2

DSP_ADCOUNT

Description

Acquisition delay counter, in samples.

See Also

DSP_DACOUNT

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_BOARDCLASS

Description

Board classification number used in DSP code.

Comments

DSP/analog hardware types are classified by number. If hardware types are close related, with only minor differences such as analog I/O characteristics, DSP_BOARDSUBCLASS may be used.

See Also

DSP_BOARDSUBCLASS

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_BOARDSUBCLASS

Description

Board subclassification number used in DSP code.

Comments

DSP/analog hardware types are normally classified by the DSP_BOARDCLASS variable. If hardware types are close related, with only minor differences such as analog I/O characteristics, DSP_BOARDSUBCLASS may be used.

See Also

DSP_BOARDCLASS

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_BUFLEN

Description

Buffer length used in acquisition, generation, instrument, or file-based acceleration processing functions.

See Also

DSP_FRMSIZ

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_BUFNUM

Description

Current DSP buffer flag in continuous analog I/O functions which use a buffer-switching mechanism.

See Also

DSP_HOSTBUFNUM

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_BUSYFLG

Description

DSP busy flag.

Comments

Used by file-based acceleration processing functions to indicate when function is busy or idle.

See Also

DSP_RIFLG

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_CHANLIST

Description

Analog input channel list.

Comments

Each trace occupies a 4-bit channel field in this value, starting with

trace 0 in bits 0..3.. For DSP/analog hardware types with 32-bit floating-point DSP devices, this normally means a maximum of 8 channels can be specified.

See Also

DSP_COUPLIST

DSP_GAINLIST

DSP_TRIGCHANLIST

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_CODECCTRL1

Description

CODEC control word 1.

Comments

Used only by DSP/analog hardware types with analog I/O CODEC devices which have special configuration settings or options not controllable by standard DSP variables sent from the DSP Engine. Examples of non-standard settings include sin(x)/x filter enable/disable, µLaw vs. linear data format, bit-order reversal, etc.

See Also

DSP_CODECCTRL2

DSP_SERIALCTRL

DSP_FSMODE

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_CODECCTRL2

Description

CODEC control word 2.

Comments

Used only by DSP/analog hardware types with analog I/O CODEC devices which have special configuration settings or options not controllable by standard DSP variables sent from the DSP Engine. Examples of non-standard settings include sin(x)/x filter enable/disable, μLaw vs. linear data format, bit-order reversal, etc.

See Also

DSP_CODECCTRL1

DSP_SERIALCTRL

DSP_FSMODE

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_COUPLIST

Description

Analog input coupling list.

Comments

Each input trace occupies a 4-bit channel field in this value, starting with trace 0 in bits 0..3. Currently defined values include 0 = dc coupling, 1 = ac coupling, 2 = microphone input, 3 = auxiliary input. For DSP/analog hardware types with 32-bit floating-point DSP devices, this normally means a maximum of 8 channels can be specified.

See Also

DSP_CHANLIST

DSP_GAINLIST

DSP_TRIGCHANLIST

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_DACOUNT

Description

Stimulus output delay counter, in samples.

See Also

DSP_ADCOUNT

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_DUPFLG

Description

Duplicate trace flag.

Comments

Set if instrument function trace 2 should be duplicated from trace 1.

See Also

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_FILTADDR1

Description

Filter coefficient address for trace 1 in acquisition, generation, and instrumentation functions if real-time filtering is enabled.

Comments

Filter coefficients are either in floating-point or quantized fixed-point form, depending on the DSP device in use. Onchip DSP memory is used, if possible, for optimal real-time performance.

See Also

DSP_FILTLEN1

DSP_FILTTYPE1

DSP_FILTQUANT

DSP_FILTUPDFLG

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_FILTADDR2

Description

Filter coefficient address for trace 2 in acquisition, generation, and instrumentation functions if real-time filtering is enabled.

Comments

Filter coefficients are either in floating-point or quantized fixed-point form, depending on the DSP device in use. Onchip DSP memory is used, if possible, for optimal real-time performance.

See Also

DSP_FILTLEN2

DSP_FILTTYPE2

DSP_FILTQUANT

DSP_FILTUPDFLG

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_FILTLEN1

Description

Length of filter for trace 1 in acquisition, generation, and instrumentation functions if real-time filtering is enabled.

Comments

Specifies number of taps for FIR filter, filter order for IIR filters. A value of 0 indicates real-time filtering for trace 1 is disabled.

See Also

DSP_FILTADDR1

DSP_FILTLEN1

DSP_FILTQUANT

DSP_FILTUPDFLG

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_FILTLEN2

Description

Length of filter for trace 2 in acquisition, generation, and instrumentation functions if real-time filtering enabled.

Comments

Specifies number of taps for FIR filter, filter order for IIR filters. A value of zero indicates real-time filtering for trace 2 is disabled.

See Also

DSP_FILTADDR2

DSP_FILTTYPE2

DSP_FILTQUANT

DSP_FILTUPDFLG

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_FILTTYPE1

Description

Type of filter for trace 1 in acquisition, generation, and instrumentation functions if real-time filtering is enabled.

Comments

Specifies FIR filter, IIR filter modified direct form, or IIR filter balanced pole-zero form.

See Also

DSP_FILTADDR1

DSP_FILTLEN1

DSP_FILTQUANT

DSP_FILTUPDFLG

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_FILTTYPE2

Description

Type of filter for trace 2 in acquisition, generation, and

instrumentation functions if real-time filtering is enabled.

Comments

Specifies FIR filter, IIR filter modified direct form, or IIR filter balanced pole-zero form. A value of −1 indicates that filter is identical to filter 1.

See Also

DSP_FILTADDR2

DSP_FILTLEN2

DSP_FILTQUANT

DSP_FILTUPDFLG

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_FILTQUANT

Description

Filter coefficient and delay storage element quantization, in bits, used in acquisition, generation, and instrumentation functions which allow real-time filtering

See Also

DSP_FILTADDR1

DSP_FILTLEN1

DSP_FILTTYPE1

DSP_FILTUPDFLG

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_FILTUPDFLG

Description

Filter coefficient update flag used in acquisition, generation, and instrumentation functions which allow real-time filtering.

Comments

Indicates that filter coefficients have changed and should be updated. Used mostly by DSP/analog hardware types which have onchip memory not accessible by the host, which requires filter coefficients to be moved into onchip memory from external memory when an update has occurred.

See Also

DSP_FILTADDR1

DSP_FILTLEN1

DSP_FILTTYPE1

DSP_FILTUPDFLG

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_FFTLEN

Description

FFT length used in FFT processing functions and spectrum analyzer function.

See Also

DSP_FFTORD

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_FFTORD

Description

FFT order used in FFT processing functions and spectrum analyzer function.

See Also

DSP_FFTLEN

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_FRMSIZ

Description

Framesize, in samples.

Comments

Used in file-based acceleration processing for DSP and math functions. Not used to control data acquisition, generation, or instrumentation buffer lengths.

See Also

DSP_BUFLEN

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_FSMODE

Description

Sampling rate clock generator mode or control register value.

Comments

Value used to initialize counter, timer, clock generator, etc. on DSP/analog hardware used to

generate sampling frequency for analog I/O functions. This value is normally calculated by DSP Engine, which takes into account discrete value, timer wordlength, or other limitations of DSP/analog hardware.

See Also

DSP_FSVALUE

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_FSVALUE

Description

Actual sampling rate value, in Hz.

Comments

Actual sampling frequency value being used after DSP_FSMODE value has been determined. DSP_FSMODE is normally calculated by the DSP Engine, which takes into account discrete value, timer wordlength, or other limitations of DSP/analog hardware. Note that for DSP/analog hardware with 16-bit memory width, this value may not correctly reflect sampling rates greater than 65535.

See Also

DSP_FSMODE

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_GAINLIST

Description

Analog input gain list.

Comments

Each trace occupies a 4-bit channel field in this value, starting with trace 0 in bits 0..3.. For DSP/analog hardware types with 32-bit floating-point DSP devices, this normally means a maximum of 8 channels can be specified.

See Also

DSP_CHANLIST

DSP_COUPLIST

DSP_TRIGCHANLIST

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_HOSTBUFNUM

Description

Current host buffer flag in continuous analog I/O functions which use a buffer-switching mechanism.

See Also

DSP_BUFNUM

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_IOMODULE

Description

I/O module indicator.

Comments

Currently only used by limited number of TMS320C25 hardware types.

See Also

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP_LOGCOEFFA

Description

Log magnitude scaling coefficient "a".

Comments

Used in spectrum analyzer function when log magnitude calculation is in effect. Applied in equation of form $Ylog(x) = a*log_{10}[Ylin(x)] + b$.

See Also

DSP_LOGCOEFFB

DSP_LOGFLG1

DSP_LOGFLG2

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_LOGCOEFFB

Description

Log magnitude scaling coefficient "b".

Comments

Used in spectrum analyzer function when log magnitude calculation is

in effect. Applied in equation of form $Ylog(x) = a*log_{10}[Ylin(x)] + b$.

See Also

DSP_LOGCOEFFA

DSP_LOGFLG1

DSP_LOGFLG2

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_LOGFLG1

Description

Log magnitude calculation request flag for trace 1.

Comments

Used by spectrum analyzer function.

See Also

DSP_LOGFLG2

DSP_LOGCOEFFA

DSP_LOGCOEFFB

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_LOGFLG2

Description

Log magnitude calculation request flag for trace 2.

Comments

Used by spectrum analyzer function.

See Also

DSP_LOGFLG1

DSP_LOGCOEFFA

DSP_LOGCOEFFB

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_MAGDATAADDR

Description

Frequency domain or wavelet domain magnitude/phase data base address.

Comments

For DSP/analog hardware with linear memory, magnitude and phase values alternate locations.

For hardware with vector memory, magnitude values are in X: memory and phase values in Y: memory.

See Also

DSP_RIDATAADDR

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_MAXVAL

Description

Maximum absolute value found in processing buffer.

See Also

DSP_MAXVALIMAG

DSP_MAXVALREAL

DSP_MINVAL

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_MAXVAL1

Description

Current maximum absolute amplitude value for trace 1 in data acquisition functions.

See Also

DSP_MAXVAL

DSP_MAXVAL2

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_MAXVAL2

Description

Current maximum absolute amplitude value for trace 2 in data acquisition function.

See Also

DSP_MAXVAL

DSP_MAXVAL1

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_MAXVALIMAG

Description

Maximum absolute value of imaginary component of complex data found in processing buffer.

See Also

DSP_MAXVAL

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_MAXVALREAL

Description

Maximum absolute value of real component of complex data found in processing buffer.

See Also

DSP_MAXVAL

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_MINVAL

Description

Minimum value found in acquisition or processing buffer.

See Also

DSP_MAXVAL

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_MONITORFLG

Description

Monitor mode enable/disable flag.

Comments

Used to control "monitor mode" in analog input functions. Enabling monitor mode causes analog input samples to looped back to analog output, with a one-sample delay, both before data acquisition is started, and during. If stimulus & response operation is in effect, input loopback automatically stops when stimulus output begins.

See Also

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_NUMCHAN

Description

Number of channels in analog I/O function.

See Also

DSP_CHANLIST

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_OFFSETIN

Description

Analog input data digital offset.

Comments

Specified in A/D samples. Value can be positive or negative and is added to analog input samples after digital scaling is applied (see DSP_SCALEIN).

See Also

DSP_SCALEIN

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_OFFSETOUT

Description

Analog output data digital offset.

Comments

Specified in D/A samples. Value can be positive or negative and is added to analog output samples after digital scaling is applied (see DSP_SCALEOUT).

See Also

DSP_SCALEOUT

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_OPMODE

Description

Operating mode.

Comments

Specifies processing or analog I/O function operating mode to

Hypersignal DSP Source Code Interface.

See Also

DSP_BOARDCLASS

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_OVERFLOW

Description

Overflow flag.

Comments

Indicates to host when overflow has occurred in processing or instrumentation function. Active only for DSP/analog hardware with fixed-point DSP devices.

See Also

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_PHZREQ

Description

Phase data calculation request flag.

Comments

Used in spectrum analyzer function.

See Also

DSP_LOGFLG1

DSP_LOGFLG2

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_PWRCOEFFA

Description

Coefficient "a" for power spectra calculation using exponential filter method.

Comments

Used in spectrum analyzer instrument function. Exponential filter is single-pole lowpass filter of form y[n] = a∗x[n] + b∗y[n–1].

See Also

DSP_PWRCOEFFB

DSP_PWRFLG1

DSP_PWRFLG2

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_PWRCOEFFB

Description

Coefficient "b" for power spectra calculation using exponential filter method.

Comments

Used in spectrum analyzer instrument function. Exponential filter is single-pole lowpass filter of form y[n] = a∗x[n] + b∗y[n–1].

See Also

DSP_PWRCOEFFA

DSP_PWRFLG1

DSP_PWRFLG2

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_PWRFLG1

Description

Power spectra calculation request flag for trace 1.

Comments

Used in spectrum analyzer instrument function.

See Also

DSP_PWRFLG2

DSP_PWRCOEFFA

DSP_PWRCOEFFB

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_PWRFLG2

Description

Power spectra calculation request flag for trace 2.

Comments

Used in spectrum analyzer instrument function.

See Also

DSP_PWRFLG1

DSP_PWRCOEFFA

DSP_PWRCOEFFB

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_RIDATAADDR

Description

Frequency domain or wavelet domain complex (real/imaginary) data base address.

Comments

For DSP/analog hardware with linear memory, real and imaginary values alternate locations. For hardware with vector memory, real values are in X: memory and imaginary values in Y: memory.

See Also

DSP_MAGDATAADDR

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_RIFLG

Description

Real/imaginary data flag.

Comments

Used to indicate that DSP/math function should also format and prepare real/imaginary data for host transfer.

See Also

DSP_RIDATAADDR

DSP_MAGDATAADDR

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_SCALEIN

Description

Analog input digital scale factor.

Comments

An integer value specified in "Q-10" notation, to allow both gain and attenuation. Typical effective scaling range is from 1/1024 to 256 is achieved by multiplying desired value by 1024. Applied before digital offset value is added (see DSP_OFFSETIN).

See Also

DSP_OFFSETIN

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_SCALEOUT

Description

Analog output data digital scaling factor.

Comments

An integer value specified in "Q-10" notation, to allow both gain and attenuation. Typical effective scaling range is from 1/1024 to 256 is achieved by multiplying desired value by 1024. Applied before digital offset value is added (see DSP_OFFSETOUT).

See Also

DSP_OFFSETOUT

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_SERIALCTRL

Description

Control word for DSP onchip serial port configuration.

Comments

DSP/analog hardware types which use onchip serial ports to interface to external A/D or D/A converters most often require a fixed serial port control word value. However, some hardware types, most notably DSP32C and DSP3210, may allow some user-specified options in this value. In these cases, this value is active.

See Also

DSP_CODECCTRL1

DSP_CODECCTRL2

DSP_FSMODE

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_STMBUFLEN

Description

Stimulus data buffer length.

Comments

Stimulus data is output using dual-buffer switching mechanism; thus, total stimulus length is twice value of DSP_STMBUFLEN.

See Also

DSP_STMDATAADDR

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_STMDATAADDR

Description

Stimulus data base address in DSP memory.

Comments

For TMS320C2x and TMS320C5x DSP/analog hardware, this address is in program memory. For all other hardware types, this address is in data memory.

See Also

DSP_STMBUFLEN

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP_TIMDATAADDR

Description

Time domain data base address in DSP memory.

Comments

Used for both file-based acceleration processing and analog I/O functions.

See Also

DSP_WINDATAADDR

DSP_RIDATAADDR

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_TRIGCHANLIST

Description

Analog input trigger channel list.

Comments

Each trace occupies a 4-bit channel field in this value, starting with trace 0 in bits 0..3.. For DSP/analog hardware types with 32-bit floating-point DSP devices, this normally means a maximum of 8 channels can be specified.

See Also

DSP_TRIGLEVEL

DSP_CHANLIST

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_TRIGLEVEL

Description

Analog input trigger level.

Comments

Specified in A/D samples; can be either positive or negative. A value of zero indicates "freerun" trigger mode.

See Also

DSP_TRIGCHANLIST

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_WINSCL

Description

Scaling factor compensating for specified FFT or wavelet transform window type, if any.

Comments

Used by spectrum analyzer function when calculating correct linear magnitude scale factors.

See Also

DSP_WINDATAADDR

DSP_LOGFLG1

DSP_LOGCOEFFA

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_WINDATAADDR

Description

Time domain window base address in DSP memory.

Comments

Used for FFT or other DSP processing function, and spectrum analyzer instrument function.

See Also

DSP_WINSCL

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_XFERFLG1

Description

Transfer function calculation request flag in spectrum analyzer function for trace 1.

See Also

DSP_XFERFLG2

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP_XFERFLG2

Description

Transfer function calculation request flag for trace 2 in spectrum analyzer instrument function.

See Also

DSP_XFERFLG1

DSGetHVarMem

DSPutHVarMem

Related Users Guide Topics

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interfaces for Hypersignal in Engine Mode

4 Waveform File Formats

Overview

[@k file format, format,, file]

This section provides a list of the various file formats supported directly and indirectly by Signalogic DSP software. There are three levels of waveform file format support:

1. Format is supported directly by software. Software functions either read format automatically by using a search based on waveform format filename extension, or require a waveform filename extension to be entered explicitly.

2. Format can be converted by utility function to a format supported directly by software. If waveform file format has a header, all values

and attributes will be converted that have counterparts in the directly supported format.

3. Format can be "imported", using a general-purpose File Acquisition or similar utility to convert file data into a format directly supported by the software. In this case, if the waveform file format has a header, it is likely that some header values and attributes may not be converted fully, or lost.

Main Topics

Hypersignal Waveform File Format

Microsoft .WAV Waveform File Format

MATLAB Waveform File Format

Acoustic/Audio Waveform File Formats

Techron .TDS File Format

Ariel SYSid File Format

Micro Technology Unlimited .SF File Format

4.1 Hypersignal Waveform File Format

Below is a quick reference for version 4.x and higher Hypersignal waveform file format. For additional, detailed information about individual values in the Hypersignal waveform file format, see Hypersignal-Macro Series Users Manual chapter 5, Hypersignal Waveform File Format.

Table 1 Hypersignal Waveform File Format Quick Reference

Name	Word[1] Index	Size / Type

SHORT HEADER		
Maximum Amplitude	0	word
Framesize	1	word
Sampling Frequency remainder[2]	2	integer (0 to 32766)
FFT Order	3 (b7-b0)	byte
Amplitude Exponent	3 (b15-b8)	byte
Waveform Number of Elements (NE) remainder	4	integer (0 to 16383)
Frame Overlap	5	word
Data Type Attributes	6	
Time Domain Source	6 (b1-b0)	2 bits
Extended Header Attribute	6 (b2)	1 bit
Complex Attribute	6 (b3)	1 bit
LPC Order	6 (b8-b4)	5 bits
Window Type	6 (b11-b9)	3 bits
Amplitude Exponent	6 (b15-b12)	4 bits
Number of Channels	7 (b5-b0)	6 bits
Nyquist FFT pt. Attribute	7 (b6)	1 bit
Autoscale Attribute	7 (b7)	1 bit
User-Defined	7 (b15-b8)	byte
Sampling Frequency quotient[2]	8	integer (0 to 32767)
Waveform Number of Elements (NE) quotient	9	integer (0 to 32767)
EXTENDED HEADER		
Minimum Amplitude	10	word
Calibration Offset	11	single
Gain1	13	single
Gain2	15	single
Data Precision	17	word
Inp Level[3]	18	single
Start Freq	20	single

Extended Framesize[4]	22	doubleword
FFT Size	24	doubleword
Extended Frame Overlap[5]	26	doubleword
Floating-Point Amplitude	28	double
Transform Window Scale Value	32	single
Reserved for Signalogic Use	**34**	**14 words**
User-Defined	**48**	**16 words**
Start of Waveform Data	**64**	**NE integers**

Notes

[1] Unless otherwise mentioned, words are 16-bit unsigned integers ranging in value from 0..65535, integers are 16-bit values ranging from -32768..32767, doublewords are 32-bit unsigned integers ranging in value from 0..4294967295, singles are IEEE format 32-bit floating-point values, and doubles are IEEE format 64-bit floating-point values.

[2] See Hypersignal-Macro Series Users Manual chapter 5, Hypersignal Waveform File Format, for explanation on Sampling Frequency remainder, quotient and Waveform length remainder and quotient

[3] The QUANT field in earlier versions of Hypersignal has been removed. Its value is now contained as part of the multiplier in the Inp Level field. The new Inp Level value is calculated as 2**(QUANT-15)

[4] Valid only when Framesize field is zero.

[5]Valid only when Frame Overlap field is zero.

General Waveform File Format Topics

Hypersignal Waveform File Format

Microsoft .WAV Waveform File Format

Additional Notes About .WAV File Limitations

MATLAB Waveform File Format

Acoustic/Audio Waveform File Formats

Techron .TDS File Format

Ariel SYSid File Format

Micro Technology Unlimited .SF File Format

4.2 Microsoft .WAV Waveform File Format

The "standard" Microsoft .WAV file format, unless modified by user-defined extension, is exclusively a time domain format. By standard, we mean .WAV files shipped with Windows 3.x and popular off-the-shelf multimedia products. The standard format contains a simple 44-byte header, which allocates no values for frequency domain information, wavelet domain information, or for other math/DSP analysis purposes. Furthermore, although the .WAV format is theoretically a subset of the general Microsoft RIFF format, and can thus be extended to contain user-defined header sections, experience at Signalogic has shown

that most off-the-shelf software packages fail to ignore sections they do not understand, and either crash or process incorrect data.

Below is a quick reference for standard .WAV file formats. See the *Microsoft Windows Multimedia Programmer's Reference*, chapter 8, for detailed information on the .WAV format. Also, see the topic Hypersignal Tim2Wav and Wav2Tim Waveform File Conversion, in Signalogic DSP Software System Guide section 1.2.1, Summary Of Hypersignal-Macro Series Software.

Table 2 Microsoft WAVE File Format Quick Reference

Name	Word[1] Index	Size / Type
SHORT HEADER		
'RIFF" section tag	0	4 bytes
Length of RIFF section	2	doubleword
'WAVE' format identifier	4	4 bytes
'fmt ' section tag	6	4 bytes
Length of fmt section	8	doubleword
PCM identifier	10	word
Number of Channels	11	integer
Sampling Frequency	12	doubleword
Average Bytes Per Second	14	doubleword
Block Alignment	16	integer
Precision (8-bit or 16-bit)	17	word
'data' section tag	18	4 bytes
Length of data section (NB)	20	doubleword
Start of Waveform Data	**22**	**NB/2 words**

Notes

[1] Unless otherwise mentioned, words are 16-bit unsigned integers ranging in value from 0..65535, integers are 16-bit values ranging from -32768..32767, doublewords are 32-bit unsigned integers ranging in value from 0..4294967295, singles are IEEE format 32-bit floating-point values, and doubles are IEEE format 64-bit floating-point values.

General Waveform File Format Topics

Hypersignal Waveform File Format

Microsoft .WAV Waveform File Format

Additional Notes About .WAV File Limitations

MATLAB Waveform File Format

Acoustic/Audio Waveform File Formats

Techron .TDS File Format

Ariel SYSid File Format

Micro Technology Unlimited .SF File Format

Additional Notes About .WAV File Limitations

There are several limitations to the de facto standard 44-byte header .WAV format found in use with most multimedia programs. One of the most noticeable is the lack of maximum amplitude value, which causes .WAV files to be displayed and processed in Hypersignal and DSPower software as if they always contained full-scale values. Another is lack of frequency or

wavelet domain information. These are severe limitations when working with Hypersignal or DSPower software, and it is strongly recommended that, at the earliest opportunity, you convert your .WAV files into Hypersignal .TIM format one of the following ways:

- Use the Wav2Tim conversion function either from the Hypersignal Macro Manger menu or as a DSPower-Block Diagram block.
- Use File Acquisition function, either from Hypersignal or as DSPower-Block Diagram block.
- Use the .WAV file as direct input into Hypersignal or DSPower-Block Diagram function, by specifying the .WAV extension explicitly in an input waveform filename field.

4.3 MATLAB Waveform File Format

Section not available yet.

General Waveform File Format Topics

Hypersignal Waveform File Format
Microsoft .WAV Waveform File Format
Additional Notes About .WAV File Limitations
MATLAB Waveform File Format
Acoustic/Audio Waveform File Formats
Techron .TDS File Format
Ariel SYSid File Format
Micro Technology Unlimited .SF File Format

4.4 Acoustic/Audio Waveform File Formats

Mainly due to the influence of Hypersignal-Acoustic, information is available for some popular acoustic/audio analysis formats in the following sections.

Acoustic/Audio Waveform File Formats

4.4.1 Techron .TDS File Format

The .TDS file format is a frequency domain ASCII (text file) format used by Techron with their popular TEF20 analyzer system and SoundLab software. A tefrd.mac function is included with Hypersignal-Acoustic software, which can convert Techron .TDS files into Hypersignal .FRQ, .PHZ,

and .REA/.IMA waveform files. These files can then be used directly in Hypersignal frequency domain functions and displays, including iFFT to produce equivalent time domain responses, and the equalize.mac function, to produce equalizing time domain responses.

The tefrd.mac function invokes the tefrd.exe utility, for which source code is included with Hypersignal-Acoustic software. Please see this source code for further information on the .TDS file format.

General Waveform File Format Topics

Hypersignal Waveform File Format

Microsoft .WAV Waveform File Format

Additional Notes About .WAV File Limitations

MATLAB Waveform File Format

Acoustic/Audio Waveform File Formats

Techron .TDS File Format

Ariel SYSid File Format

Micro Technology Unlimited .SF File Format

4.4.2 Ariel SYSid File Format

The .DAT file format is a frequency domain ASCII (text file) format used by Ariel with their popular SYSid analyzer system and software. A sysidrd.mac function is included with Hypersignal-Acoustic

software, which can convert Ariel .DAT files into Hypersignal .FRQ, .PHZ, and .REA/.IMA waveform files. These files can then be used directly in Hypersignal frequency domain functions and displays, including iFFT to produce equivalent time domain responses, and the equalize.mac function, to produce equalizing time domain responses.

The sysidrd.mac function invokes the sysidrd.exe utility, for which source code is included with Hypersignal-Acoustic software. Please see this source code for further information on SYSid file formats.

General Waveform File Format Topics

Hypersignal Waveform File Format

Microsoft .WAV Waveform File Format

Additional Notes About .WAV File Limitations

MATLAB Waveform File Format

Acoustic/Audio Waveform File Formats

Techron .TDS File Format

Ariel SYSid File Format

Micro Technology Unlimited .SF File Format

4.4.3 Micro Technology Unlimited .SF File Format

Micro Technology Unlimited (MTU) .SF format waveform files

can be imported into Hypersignal-Macro series software or DSPower software formats using the File Acquisition function. Use the following settings:

- format set to 'BIN'
- sample width set to 4 (16-bit)
- bytes to discard set to 1024 for mono or stereo waveform input, or set to 1026 to import only channel B from a stereo waveform file
- number of samples to acquire set to 0 (all)
- sampling frequency set to appropriate value
- channel strip set to 0 for mono or stereo waveform input, or set to 1 to import either only channel A or channel B from a stereo waveform file
- number of channels set to 1 for mono acquisition, or 2 for stereo acquisition

General Waveform File Format Topics

Hypersignal Waveform File Format

Microsoft .WAV Waveform File Format

Additional Notes About .WAV File Limitations

MATLAB Waveform File Format

Acoustic/Audio Waveform File Formats

Techron .TDS File Format

Ariel SYSid File Format

Micro Technology Unlimited .SF File Format

5 Definitions

Overview

This section of the Signalogic DSP Software Reference Guide documents structures defined for use in application-specific C/C++ and MATLAB programs when calling functions in DSPower series DLLs. The actual (and current) definition of these structures can be found in the include files noted for each. Example include hwlib.h, hwmgr.h, etc.

Structures Used in DSPower Series Software

HWLISTENTRY

CONVERSIONINFO

ERRORMSGINFO

DSPDATARDYMSGINFO

5.1 Structure Definitions

There are several structure definitions used by DSPower series

software. These are typically found in include files such as enmgr.h, hwmgr.h, hwlib.h, and splib.h. The sections below describe these structure definitions.

5.1.1 HWLISTENTRY Structure

Below is a description of the HWLISTENTRY structure, which is used in Hardware Manager calls such as DSGetHWMgrEntry and DSSetHWMgrEntry. The current definition of this structure is located in the DSPower-HWLib include file **hwmgr.h**.

The description below is divided into the following sections:

- information maintained in the Hardware Setup File
- information maintained internally by Hardware Manager, and is not stored in the Hardware Setup File

Each element in the structure description below includes a brief comment. See also the notes after the structure definition, and Signalogic DSP Software Users Guide section 8.3.1, Hardware Setup File Fields.

```
typedef struct {

                                                    // comment


                                                    //
```

manufacturerfield

//description field: name/model, DSP type, etc.

//board designatorfield

//major DSP/analogwar

e
driver
ID
field

//
minor
DSP/
analog
hardware
driver
ID
field

//
DSP
program
file
field
(current
executable
code
file
name)

//
default
DSP
pr

ogramfile

//currentI/Obaseaddressfield(inhex)

//defaultI/Obaseaddress

//currentmemorybaseadres

s
fie
ld
(in
he
x)

//
de
fa
ult
ba
se
m
e
m
or
y
ad
dr
es
s

//
bu
s
ty
pe
fie
ld

//
de
fa
ult
bu
s
ty
pe

//
pr
oc
es
so
r
clo
ck
fie
ld
(s
pe
cif
ie
d
in
M

Hz
)

//
de
fa
ult
pr
oc
es
so
r
clo
ck

//
vo
lta
ge
ra
ng
es
fie
ld
(s
ho
ul
d
be
en
ter
ed
as
'in
pu
t,
ou
tp
ut'
)

//
de
fa
ult
vo
lta
ge
ra
ng
es

//
an
al
og
clo
ck
ty
pe
fie
ld:
0
=
ex
ter
na
l
clo
ck
, 1
=
fix
ed
int
er
na
l,
2
=
pr
og
int
er
na
l

//
de
fa
ult
an
al
og
clo
ck
ty
pe

//
m
e
m
or
y
ar
ch
ite
ct
ur
e
fie
ld:

"L
"
=
lin
ea
r,
"H
"
=
m
od
ifi
ed
Ha
rv
ar
d,
"V
"
=
ve
ct
or

//
de
fa
ult
m
e
m
or
y
ar
ch
ite
ct
ur
e

//
nu
m
be
r
of
pr
oc
es
so
rs
fie
ld

//
de
fa
ult
nu
m
be
r
of

```
    processors

    // following parameters not stored in Hardware Setup File

                                                    // comment

                                                    // calls-allowed classification

                                                    // internal information used to mark pending updates, deletes, etc.
} HWLISTENTRY;

typedef HWLISTENTRY FAR* PHWLISTENTRY;
```

Notes

1) HWLISTENTRY structures should be completely initialized to zero before using, so default values can take effect for parameters not

used. This can easily be accomplished using a call such as

```
_fmemset(&HWListEntry, 0, sizeof(HWLISTENTRY));
```

2) Supporting type definitions used above:

```
typedef char DESCSTR[50];
typedef char VALSTR[10];
```

Example

See Also

DSSetHWMgrEntry

DSGetHWMgrEntry

Related Topics

Hardware Setup File Access

Selecting DSP/Analog Hardware Types

DSP/Analog Hardware Configuration

Hardware Values and Variables

General Topics

Hardware Manager Access

Hardware Selector

Hardware Setup File

Error Handling in Hardware Manager

HWLISTENTRY Example

Below is an excerpt from the dscope.cpp source code example file included with DSPower-HWLib software. It shows a typical include file section which

allows compilation of applications using the DSPower-HWLib software. Remember that referring to the prototypes inside the DSPower-HWLib DLL include files shown below will cause the necessary .lib (library) files to be needed in the application's project file at link-time; for more information on DSPower-HWLib and DSPower-SPLib .lib files, see section 4.2.1, .LIB Files for DSPower-HWLib and DSPower-SPLib.

```
// typical application-specific include files
#include <windows.h>
#include <stdlib.h>
#include <math.h>

// DSPower-HWLib DLL include files
#include "enmgr.h"   // Engine Manager
#include "hwmgr.h"   // Hardware Manager
#include "hwlib.h"   // Hardware Library

// more Signalogic DLL include files, but used only for user-interface in demo
#include "toollib.h"
#include "helplib.h"
#include "minmax.h"  // local versions of min/max, if needed
```

See Also

DSSetHWMgrEntry

DSGetHWMgrEntry

Related Topics

Hardware Setup File Access

Selecting DSP/Analog Hardware Types

DSP/Analog Hardware Configuration

Hardware Values and Variables

General Topics

Hardware Manager Access

5.1.2 CONVERSIONINFO Structure

Below is a description of the CONVERSIONINFO structure, which is used in high-level DSPower-HWLib calls such as DSAcquireWvfrmFile and DSGenerateWvfrmFile. The current definition of this structure is located in the DSPower-HWLib include file **hwlib.h**.

The reprint below is partially divided into the following sections:

- basic acquisition/generation information
- real-time digital filter information
- trigger information
- stimulus & response (simultaneous input/output information)

Each element in the structure description below includes a brief comment. For more detailed information, see the additional explanations of some elements after the structure.

```
typedef struct {

// basic acquisition/generation
information
```

/ / real-time digital filter information

/ / trigger information

/ / stimulus & response information

```
} CONVERSIONINFO;

typedef CONVERSIONINFO FAR*
PCONVERSIONINFO;
```

Notes

1) The entire CONVERSIONINFO structure should be initialized to zero before using, so that default values can take effect for parameters not used. This can easily be accomplished using a call such as

```
_fmemset(&ConversionInfo, 0, sizeof(CONVERSIONINFO));
```

2) Previous value used in either DSEngineOpen or DSSetEngineWaveformPath call is

used for waveform filenames, unless explicit path information is found in the filename above.

NOTE: For operation when a Hypersignal-Macro series program is the DSP Engine, szFilename above should not contain path information. The path should instead be set using the initial DSEngineOpen call, or a subsequent DSSetEngineWaveformPath call.

---

3) The szGainList parameter is a string because input (or output) gain and/or attenuation can be specified as a software digital scaling value, or as a hardware gain value in Volts or dB, depending on the DSP/analog hardware type. Also, entry can be channel-specific, or specified as a channel-range.

4) TrigDelay parameter not yet operational.

5) PCONVERSIONINFO can be used to declare pointers to CONVERSIONINFO structures. For examples, see Example C/C++ Code: Playing Back Waveform Files, and Example C/C++ Code: Acquiring Waveform Files, in Signalogic DSP Software Users Guide section 4.2.2, Example Code: DSPower-HWLib and DSPower-SPLib Software.

Example

See Also

DSAcquireWvfrmFile

DSGenerateWvfrmFile

Related Topics

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

CONVERSIONINFO Example

Below is a partial excerpt from the dtape.cpp example source code file included with DSPower-HWLib software showing typical setup and initialization of a CONVERSIONINFO structure for use in a DSAcquireWvfrmFile call. The dtape.cpp demo attempts to open Hypersignal-Macro or Hypersignal-Acoustic software as a DSP Engine, but the source code example below applies regardless of the engine. One exception may occur in the szFilename parameter of the CONVERSIONINFO structure; for more information, see notes in section 5.1.2 above, CONVERSIONINFO Structure.

```
// create CONVERSIONINFO object memory; use zero-initialization to set most
parameters to default values

hConvInfo = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE |
LMEM_DISCARDABLE, sizeof(CONVERSIONINFO));

pConvInfo = (PCONVERSIONINFO)LocalLock(hConvInfo);

lstrcpy(pConvInfo->szFilename, szWaveformFilename);  // file to acquire

// setup for one channel, init channel list (example is sequential)

pConvInfo->NumChan = 1;          // one channel
for (j=0; j<pConvInfo->NumChan; j++) pConvInfo->ChanList[j] = j;

// setup up sampling rate, amount, gain, trigger information

pConvInfo->Fs =22050;  // 22.05 kHz sampling rate
pConvInfo->NumSamples = 100000L;  // 100,000 samples
lstrcpy(pConvInfo->szGainList, "1");  // see notes in CONVERSIONINFO
pConvInfo->NumTrigChan = 1;  // one trigger channel
pConvInfo->TrigChanList[0] = 0;  // freerun trigger

// start acquisition

DSAcquireWvfrmFile(hBoard, pConvInfo);

// free and delete memory

LocalUnlock(hConvInfo);
LocalFree(hConvInfo);
```

See Also

DSAcquireWvfrmFile

DSGenerateWvfrmFile

Related Topics

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

5.1.3 DSPDATARDYMSGINFO Structure

Below is a description of the DSPDATARDYMSGINFO structure, which pointed to by the lParam parameter of DSP Engine callback messages such as WM_DSPENGINE_BUFRDY and WM_DSPENGINE_FLGRDY sent by Engine Manager. The current definition of this structure is located in the DSPower-HWLib include file **enmgr.h**.

Each element in the structure description below includes a brief comment. See also the notes below, after the structure definition, and Signalogic DSP Software Users Guide section Buffer Ready Messages from DSP Engines and Semaphore Messages from DSP Engines, under section 7.4, Registering Callback Messages from DSP Engines.

```
typedef struct {

                                                    //
                                                    co
                                                    mm
                                                    ent
```

```
    //
    buff
    er
    or
    flag
    nu
    mbe
    r

    //
    flag
    valu
    e

    //
    flag
    add
    ress

    //
    boa
    rd
    han
    dle

} DSPDATARDYMSGINFO;

typedef DSPDATARDYMSGINFO FAR*
PDSPDATARDYMSGINFO;
```

Notes

1) The buffer or flag number reflects the original nBufferNum or nFlagNum values in used in DSWaitForBuffer or DSWaitForFlag calls.

2) The lParam value of the WM_DSPENGINE_BUFRDY and WM_DSPENGINE_FLGRDY

messages points to a DSPDATARDYMSGINFO structure.

3) The wParam value of the WM_DSPENGINE_BUFRDY and WM_DSPENGINE_FLGRDY messages is the handle of the engine to which the DSP board buffer or flag was assigned.

Example

See Also

DSWaitForBuffer

DSWaitForFlag

DSRegisterEngineMsgWnd

WM_DSPENGINE_BUFRDY

WM_DSPENGINE_FLGRDY

Related Users Guide Topics

Registering Callback Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

DSPDATARDYMSGINFO Example

Below is an excerpt from the dscope.cpp source code example file (included with DSPower-HWLib software) showing a WM_DSPENGINE_BUFRDY message being received:

```
case WM_DSPENGINE_BUFRDY:  // buffer ready message sent by DSP engine
```

```
pBuf = (int far*)GlobalLock(hBuf);  // lock memory to be used

// transfer below illustrates two common type of DSP memory architectures

if (uMemArch == DS_GMA_VECTOR) {  // vector data memory

     if (nCurBuf == 0)  status = DSGetMem(hBoard,
     DS_GM_VECTOR_DATA_X, dwTimBufferBaseAddr, DS_GM_SIZE16,
     pBuf, wBuflen);

     else  status = DSGetMem(hBoard, DS_GM_VECTOR_DATA_Y,
     dwTimBufferBaseAddr, DS_GM_SIZE16, pBuf, wBuflen);
}
else { // linear data memory

     status = DSGetMem(hBoard, DS_GM_LINEAR_DATA_RT,
     TimBufferBaseAddr+nCurBuf*Buflen, DS_GM_SIZE16, pBuf, wBuflen);
}

GlobalUnlock(hBuf);  // let memory float

if (!status) MessageBox(NULL, "DSGetMem problem", "Test Prog",
MB_OK);

// switch buffers and write new buffer number

nCurBuf ^= 1;
DSPutEngineVarMem(hBoard, DSP_HOSTBUFNUM, nCurBuf);

// can we have another please, Sir..
DSWaitForBuffer(hBoard, nCurBuf, NULL, DS_WFB_POLLED);
break;
```

Notes

1) A great deal of initialization is necessary before buffer ready and semaphore messages can be processed. The example above leaves out much of this; it is strongly recommended that the entire dscope.cpp file be examined before implementing code similar to the above example.

2) uMemArch is assumed to have been initialized beforehand using the DSGetMemArch call.

3) hBuf is assumed to have been created via a GlobalAlloc call, using a sizeof parameter equal to 2*wBuflen. Memory pointers passed to DSGetMem and DSPutMem calls *must not point to memory segments less than the product of the dwCount and DS_GM_SIZExx parameters* used in the corresponding DSGetMem or DSPutMem calls. This product is in bytes.

See Also

DSWaitForBuffer

DSWaitForFlag

DSRegisterEngineMsgWnd

WM_DSPENGINE_BUFRDY

WM_DSPENGINE_FLGRDY

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

5.1.4 ERRORMSGINFO Structure

Below is a description of the ERRORMSGINFO structure, which is pointed to by the lParam parameter in DSP Engine status messages such as WM_DSPENGINE_COMPILEERROR and WM_DSPENGINE_RUNTIMEERROR sent by Engine Manager. The current definition of this structure is

located in the DSPower-HWLib include file **enmgr.h**.

Each element in the structure description below includes a brief comment. See also the notes below, after the structure definition, and Signalogic DSP Software Users Guide section Status Messages from DSP Engines, under section 7.4, Registering Callback Messages from DSP Engines.

```
typedef struct {

                                        // comment

                                        // error number or type

                                        // error/status string or error log filename

} ERRORMSGINFO;
```

```
typedef ERRORMSGINFO FAR*
PERRORMSGINFO;
```

Notes

1) The nErrorNum parameter can be interpreted as shown below:

Value	Meaning
>= 0	error number; szErrorStatus contains error condition string
< 0	indicates szEngineError contains pathname of error log file

2) Error log files are typically used in cases where the engine has found more than one error or warning.

3) The lParam value of the WM_DSPENGINE_xxxxERR messages points to an ERRORMSGINFO structure.

4) The wParam value of the WM_DSPENGINE_xxxxERR messages is the handle of the engine encountering the error.

Example

See Also

DSRegisterEngineMsgWnd

WM_DSPENGINE_CODEGENERROR

WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_FUNCTIONERROR

WM_DSPENGINE_RUNTIMEERROR

Related Users Guide Topics

Registering Callback Messages from DSP Engines

Status Messages from DSP Engines

ERRORMSGINFO Example

Below is a partial source code example showing a WM_DSPENGINE_COMPILEERROR message being received:

```
char szErrorStatus[256];
char szLogFile[256];
HFILE hFile;
OFSTRUCT ofstruct;

case WM_DSPENGINE_COMPILEERROR:

    int nErrorNum = (PERRORMSGINFO)lParam->nErrorNum;
    if (nErrorNum >= 0) {

        _fstrcpy(szErrorStatus, (PERRORMSGINFO)lParam->szErrorStatus);

        _fstrcat(szErrorStatus, "; Error Number = ");
        char tmpstr[30];
        itoa(nErrorNum, tmpstr, 10);
        _fstrcat(szErrorStatus, tmpstr);

        MessageBox(NULL, szErrorStatus, "Compile Error Message", MB_OK);
    }
    else {

        _fstrcpy(szLogFile, PERRORMSGINFO)lParam->szErrorStatus);
        hFile = OpenFile(szLogFile, &ofstruct);

        // process error log file...
    }

    break;
```

Note if multiple engines are open, the wParam parameter can be used to identify which engine has sent the error message.

See Also

DSRegisterEngineMsgWnd

WM_DSPENGINE_CODEGENERROR

WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_FUNCTIONERROR

WM_DSPENGINE_RUNTIMEERROR

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Status Messages from DSP Engines

5.2 Constant Definitions

There are numerous constants defined in DSPower series software include files, such as enmgr.h, hwmgr.h, hwlib.h, and splib.h. The sections below describe these constant definitions.

6 Messages

Overview

This section provides a technical description of messages which can be sent and received when using DSPower series software.

Messages have been divided into two categories:

- library messages
- DSP Engine messages

Library messages are generated by DSPower series libraries (DLLs, VBXs, OCXs, etc.) and sent to windows or controls owned by calling applications or by DSPower-Block Diagram software. These messages are essentially relays, or "reformats", of information contained in messages received from DSP Engines. Currently, messages are not sent from calling applications to DSPower series libraries; instead, various functions available in the library APIs cause the libraries to generate and send messages to DSP Engines. Note that to receive messages of any type from DSPower series libraries, calling applications must first "register" one or more window or control handles using the DSRegisterEngineMsgWnd function in Engine Manager.

By contrast, messages involving DSP Engines are transmitted directly, in both directions, between DSP Engines and other applications. Because different DSP Engines have different means of communicating with the outside world, the Windows library approach has been chosen to provide an intermediate layer and present a standardized programming interface. For

example, Hypersignal-Macro series software sends information to a Windows virtual device driver (VxD), called Pipe.386, which arbitrates bi-directional shared memory command/data queues. Messages are formed as command and data information is extracted from these queues. As another example, MATLAB programs can send information using direct Windows API calls, via external library calls, or using a method based on the built-in MATLAB interactive control interface.

Thus, in DSPower series software, DSP Engine messages are normally processed automatically, without user-intervention. From the user's point of view--a C/C++ or MATLAB API in the case of DSPower series libraries, or a block diagram interface in the case of DSPower-Block Diagram software--these messages are not visible. However, DSP Engine messages have been fully documented to allow direct communication between user-defined applications and DSP Engines, if desired.

For general information on the rationale and benefits of the DSP Engine concept, see Signalogic DSP Software System Guide section 1.1, Relationship of Signalogic DSP Software Components.

Main Message Categories

Library Messages

6.1 Library Messages

The DSPower series library messages defined below can be received by calling applications which have implicitly or explicitly loaded Engine Manager (included with DSPower-HWLib and DSPower-SPLib software; see Signalogic DSP Software Users Guide section 7.1, Engine Manager Access). There are some important items to note about DSPower series librarymessages:

Calling applications must first "register" one or more window or control handles using the DSRegisterEngineMsgWnd function in Engine Manager, before receiving these messages.

All library messages are standard Windows "WM_" messages (declared using the WM_USER constant), and not Windows notification messages (signified, for example, by "EN_", "CBN_", etc.). Notification messages are not used because all messages

contain one or more pieces of data.

All library messages begin with "WM_DSPENGINE_".

Library Messages

WM_DSPENGINE_CODEGENERROR

WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_BUFNUM

WM_DSPENGINE_BUFSIZE

WM_DSPENGINE_BUFRDY

WM_DSPENGINE_ENGINESTATE

WM_DSPENGINE_FLGRDY

WM_DSPENGINE_FUNCTIONERROR

WM_DSPENGINE_LEVEL

WM_DSPENGINE_PROGSTATUS

WM_DSPENGINE_RUNTIMEERR

WM_DSPENGINE_SAMPFREQ

WM_DSPENGINE_CODEGENERROR

WM_DSPENGINE_CODEGENERROR

```
hEngine = (HENGINE) wParam;          // engine identifier
pErrorMsgInfo = (PERRORMSGINFO)      // pointer to error information
lParam;
```

The WM_DSPENGINE_CODEGENERROR engine status message is sent by Engine Manager after receiving an error message from a

DSP Engine which has encountered a format error or an in error in locating a native language source code file or executable file.

Parameter	Description
hEngine	handle of DSP Engine encountering code generation error
pErrorMsgInfo	pointer to PERRORMSGINFO structure, which contains additional error information

Returns

The calling application does not need to return a specific value.

Comments

The pErrorMsgInfo parameter points to an ERRORMSGINFO structure, which can be summarized as follows:

```
// comment

// error number or type

// error/status string or error log
```

If nErrorNum is positive, then it represents a DSP Engine native language source or executable file processing error number (as defined in DSP Engine documentation) and szStatusString is an error condition string. If the nErrorNum parameter is negative, then szStatusString is a pathname for an error log file. Currently, for DSPower series software-compatible engines, error log files are only produced during compilation (i.e. currently, negative nErrorNum values only apply to WM_DSPENGINE_COMPILEERROR messages). For more information, see section 5.1.4, ERRORMSGINFO Structure.

C/C++ Example

Below is a partial source code example showing a WM_DSPENGINE_CODEGENERROR message being received:

```
char szErrorStatus[256];
char szLogFile[256];
HFILE hFile;
OFSTRUCT ofstruct;

case WM_DSPENGINE_CODEGENRROR:

    nErrorNum = (PERRORMSGINFO)lParam->nErrorNum;
    if (nErrorNum >= 0) {

        _fstrcpy(szErrorStatus, (PERRORMSGINFO)lParam->szErrorStatus);

        _fstrcat(szErrorStatus, "; Error Number = ");
```

```
        char tmpstr[30];
        itoa(nErrorNum, tmpstr, 10);
        _fstrcat(szErrorStatus, tmpstr);

        MessageBox(NULL, szErrorStatus, "Code Generation Error Message",
        MB_OK);
    }
    else {

        _fstrcpy(szLogFile, PERRORMSGINFO)lParam->szErrorStatus);
        hFile = OpenFile(szLogFile, &ofstruct);

        // process error log file...
    }

    break;
```

Note if multiple engines are open, the wParam parameter can be used to identify which engine has sent the error message.

See Also

WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_FUNCTIONERROR

WM_DSPENGINE_RUNTIMEERROR

DSRegisterEngineMsgWnd

DSExecEngineFunction

Related Users Guide Topics

Status Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_COMPILEERROR

```
hEngine = (HENGINE) wParam;          // engine identifier
pErrorMsgInfo = (PERRORMSGINFO)      // pointer to error information
lParam;
```

The WM_DSPENGINE_COMPILEERROR engine status message is sent by Engine Manager after receiving an error message from a DSP Engine which has encountered an error compiling a native language source code file.

Parameter	Description
hEngine	handle of DSP Engine encountering compiler error
pErrorMsgInfo	pointer to PERRORMSGINFO structure, which contains additional error information

Returns

The calling application does not need to return a specific value.

Comments

The pErrorMsgInfo parameter points to an ERRORMSGINFO structure, which can be summarized as follows:

```
// comment

// error number or type

// erro
```

r/st
atus
stri
ng
or
erro
r
log
filen
ame

If nErrorNum is positive, then it represents a DSP Engine compilation error number (as defined in DSP Engine documentation) and szStatusString is an error condition string. If the nErrorNum parameter is negative, then szStatusString is a pathname for an error log file. For more information, see section 5.1.4, ERRORMSGINFO Structure.

C/C++ Example

Below is a partial source code example showing a WM_DSPENGINE_COMPILEERROR message being received:

```
char szErrorStatus[256];
char szLogFile[256];
HFILE hFile;
OFSTRUCT ofstruct;

case WM_DSPENGINE_COMPILEERROR:

    nErrorNum = (PERRORMSGINFO)lParam->nErrorNum;
    if (nErrorNum >= 0) {

        _fstrcpy(szErrorStatus, (PERRORMSGINFO)lParam->szErrorStatus);

        _fstrcat(szErrorStatus, "; Error Number = ");
        char tmpstr[30];
```

```
        itoa(nErrorNum, tmpstr, 10);
        _fstrcat(szErrorStatus, tmpstr);

        MessageBox(NULL, szErrorStatus, "Compile Error Message", MB_OK);
    }
    else {

        _fstrcpy(szLogFile, PERRORMSGINFO)lParam->szErrorStatus);
        hFile = OpenFile(szLogFile, &ofstruct);

        // process error log file...
    }

    break;
```

Note if multiple engines are open, the wParam parameter can be used to identify which engine has sent the error message.

See Also

WM_DSPENGINE_CODEGENERROR

WM_DSPENGINE_FUNCTIONERROR

WM_DSPENGINE_RUNTIMEERROR

DSRegisterEngineMsgWnd

DSExecEngineFunction

Related Users Guide Topics

Status Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_BUFNUM

WM_DSPENGINE_BUFNUM

```
hEngine = (HENGINE) wParam;     // engine identifier
dwBufNum = (DWORD) lParam;      // buffer number
```

The WM_DSPENGINE_BUFNUM engine update message is sent by Engine Manager after receiving a buffer number message from an engine executing a high-level data acquisition or generation function. The lParam parameter contains the current buffer number of the high-level generation function.

Parameter	Description
hEngine	handle of engine sending buffer number message
dwBufNum	current buffer number of high-level data acquisition or generation function

Returns

The calling application does not need to return a specific value.

Comments

The dwBufNum parameter indicates the current buffer number of the high-level data acquisition or generation function. Buffer numbers range from 0 to the maximum 32-bit word – 1, and increase monotonically at a constant rate dependent on the sampling frequency and buffer size being used in the high-level function. The current number of samples acquired (or generated) can be calculated by multiplying dwBufNum by the result obtained previously from a WM_DSPENGINE_BUFSIZE message.

C/C++ Example

Below is an example which shows a WM_DSPENGINE_BUFNUM message being processed:

```
DWORD dwBufSize, dwBufNum;
float Fs;

case WM_DSPENGINE_SAMPFREQ:

    _fmemcpy(&Fs, &lParam, sizeof(float));  // store actual Fs being used
    break;

case WM_DSPENGINE_BUFSIZE:

    dwBufSize = lParam;  // store the buffer length being used
    break;

case WM_DSPENGINE_BUFNUM:

    dwCurSamplePos = dwBufSize*lParam;  // calculate current sample count

    char tmpstr[30], tmpstr2[30];
    ltoa(dwCurSamplePos, tmpstr, 10);  // convert to text
    ftoa(dwCurSamplePos/Fs, tmpstr2, 10);

    UpdateStatusBar(hSampleCounter, tmpstr, "Current Sample Count");
    UpdateStatusBar(hTimeCounter, tmpstr2, "Current Time (seconds)");
    break;
```

See Also

WM_DSPENGINE_BUFSIZE

WM_DSPENGINE_SAMPFREQ

WM_DSPENGINE_LEVEL

DSAcquireWvfrmFile

DSGenerateWvfrmFile

DSExecEngineFunction

Related Users Guide Topics

Update Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 4.5 Acquiring Data Using DSPower-HWLib Software

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_BUFSIZE

WM_DSPENGINE_BUFSIZE

```
hEngine = (HENGINE) wParam;      // engine identifier
dwBufSize = (DWORD) lParam;      // buffer size
```

The WM_DSPENGINE_BUFSIZE engine update message is sent by Engine Manager after receiving a buffer size message from a DSP Engine executing a high-level data acquisition, generation, or instrument function.

Parameter	Description
hEngine	handle of engine sending buffer size message
dwBufSize	buffer size being used during high-level data acquisition, generation or instrument function

Returns

The calling application does not need to return a specific value.

Comments

The dwBufSize parameter indicates the buffer size being used during the high-level data acquisition, generation, or instrument function. Buffer sizes typically range from a few hundred samples up to 65535 samples.

Once the DSP Engine has calculated an appropriate buffer size, taking into account factors such as sampling rate, number of channels, and continuous signal

generation requirements, the resulting value does not change for the duration of the high-level function. Note that in the case of continuous disk file acquisition or generation functions, it is typical for DSP Engines to calculate larger buffer sizes for high sampling rate values, in order to improve performance.

C/C++ Example

Below is an example which shows a WM_DSPENGINE_BUFSIZE message being processed:

```
DWORD dwBufSize;
float Fs;

case WM_DSPENGINE_SAMPFREQ:

    _fmemcpy(&Fs, &lParam, sizeof(float));  // store actual Fs being used
    break;

case WM_DSPENGINE_BUFSIZE:

    dwBufSize = lParam;  // store the buffer length being used

    char tmpstr[30];
    ftoa(dwBufSize/Fs, tmpstr, 10);

    UpdateStatusBar(hTimeBase, tmpstr, "Framesize (in seconds)");
    break;
```

See Also

WM_DSPENGINE_BUFNUM

WM_DSPENGINE_SAMPFREQ

WM_DSPENGINE_LEVEL

DSAcquireWvfrmFile

DSGenerateWvfrmFile

DSExecEngineFunction

Related Users Guide Topics

Update Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 4.5 Acquiring Data Using DSPower-HWLib Software

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_BUFRDY

WM_DSPENGINE_BUFRDY

```
hEngine = (HENGINE) wParam;                    // engine identifier
pDSPDataRdyMsgInfo =                           // pointer to buffer information
(PDSPDATARDYMSGINFO) lParam;
```

The WM_DSPENGINE_BUFRDY engine callback message is sent by Engine Manager after receiving a buffer ready message from a DSP Engine which has acquired, generated, or processed a DSP/analog hardware data buffer.

Parameter	Description
hEngine	handle of DSP Engine encountering compiler error
pDSPDataRdyMsgInfo	pointer to PDSPDATARDYMSGINFO structure, which contains additional buffer information

Returns

The calling application does not need to return a specific value.

Comments

pDSPDataRdyMsgInfo points to a DSPDATARDYMSGINFO structure, which can be summarized as follows:

```
//
co
```

mm
ent

//
buff
er
or
flag
nu
mbe
r

//
flag
valu
e

//
flag
add
ress

//
boa
rd
han
dle

For more information, see section 5.1.3, DSPDATARDYMSGINFO Structure.

C/C++ Example

Below is an excerpt from the dscope.cpp source code example file (included with DSPower-

HWLib software) showing a WM_DSPENGINE_BUFRDY message being received:

```
case WM_DSPENGINE_BUFRDY:  // buffer ready message sent by DSP engine

    pBuf = (int far*)GlobalLock(hBuf);  // lock memory to be used

    // transfer below illustrates two common type of DSP memory architectures

    if (uMemArch == DS_GMA_VECTOR) {  // vector data memory

        if (nCurBuf == 0)  uStatus = DSGetMem(hBoard,
        DS_GM_VECTOR_DATA_X, dwTimBufferBaseAddr, DS_GM_SIZE16,
        pBuf, wBuflen);

        else  uStatus = DSGetMem(hBoard, DS_GM_VECTOR_DATA_Y,
        dwTimBufferBaseAddr, DS_GM_SIZE16, pBuf, wBuflen);
    }
    else { // linear data memory

        uStatus = DSGetMem(hBoard, DS_GM_LINEAR_DATA_RT,
        dwTimBufferBaseAddr+nCurBuf*Buflen, DS_GM_SIZE16, pBuf,
        wBuflen);
    }

    GlobalUnlock(hBuf);  // let memory float

    if (!uStatus) MessageBox(NULL, "DSGetMem problem", "Test Prog",
    MB_OK);

    // switch buffers and write new buffer number

    nCurBuf ^= 1;
    DSPutHVarMem(hBoard, DSP_HOSTBUFNUM, nCurBuf);

    // can we have another please, Ma'am..
    DSWaitForBuffer(hBoard, nCurBuf, NULL, DS_WFB_POLLED);
    break;
```

A significant amount of initialization is necessary before buffer ready and semaphore messages can be processed. The example above leaves out much of this; it is strongly recommended that the entire dscope.cpp file be examined before implementing code similar to the above example.

For example, uMemArch is assumed to have been initialized beforehand using the DSGetMemArch call.

In the above example, hBuf is assumed to have been created via a GlobalAlloc call, using a sizeof parameter equal to 2∗wBuflen. Memory pointers passed to DSGetMem and DSPutMem calls *must not point to memory segments less than the product of the dwCount and DS_GM_SIZExx parameters* used in the corresponding DSGetMem or DSPutMem calls. This product is in bytes.

See Also

WM_DSPENGINE_FLGRDY

DSWaitForBuffer

DSWaitForFlag

DSRegisterEngineMsgWnd

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

WM_DSPENGINE_ENGINESTATE

WM_DSPENGINE_ENGINESTATE

```
hEngine = (HENGINE) wParam;          // engine identifier
wEngineState = LOWORD(lParam);       // current engine state
```

The WM_DSPENGINE_ENGINESTATE engine status message is sent by Engine Manager after receiving a status message from a DSP Engine indicating its current state, or when Engine Manager suspends or resumes an engine.

Parameter	Description
hEngine	handle of engine for which engine state message applies
wEngineState	current state of engine, as defined by one of the values below:

Value	Meaning
DS_ES_RUNNING	engine is in running (busy) state
DS_ES_IDLE	engine is in idle state
DS_ES_STOPPENDING	engine is running, but a stop (shutdown) command is pending
DS_ES_STOPPED	engine is stopped (shutdown)

Returns

The calling application does not need to return a specific value.

Comments

The wEngineState parameter reflects the current state of the DSP Engine. Note that receiving a WM_DSPENGINE_ENGINESTATE message does not necessarily mean there has been a state change. To detect changes, the calling application should update a local variable as part of its WM_DSPENGINE_ENGINESTATE message processing.

C/C++ Example

An example of processing a WM_DSPENGINE_ENGINESTATE message is shown below:

```
WORD static wPrevEngineState = DS_ES_IDLE;
WORD wEngineState;

case WM_DSPENGINE_ENGINESTATE:

    wEngineState = LOWORD(lParam);

    if (wEngineState == DS_ES_IDLE && wEngineState != wPrevEngineState) {

        UpdateStatusBar(hState, "Idle");  // engine state returned to idle
    }
    wPrevEngineState = wEngineState;  // update local copy of engine state
    break;
```

See Also

WM_DSPENGINE_PROGSTATUS

DSExecEngineFunction

Related Users Guide Topics

Status Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_FLGRDY

WM_DSPENGINE_FLGRDY

```
hEngine = (HENGINE) wParam;                    // engine identifier
pDSPDataRdyInfo =
(PDSPDATARDYMSGINFO) lParam;                   // pointer to flag information
```

The WM_DSPENGINE_FLGRDY engine callback message is sent by Engine Manager after receiving a buffer ready message from a DSP Engine which has acquired,

generated, or processed a DSP/analog hardware data buffer.

Parameter	Description
hEngine	handle of DSP Engine encountering compiler error
pDSPDataRdyInfo	pointer to PDSPDATARDYMSGINFO structure, which contains additional flag information

Returns

The calling application does not need to return a specific value.

Comments

pDSPDataRdyMsgInfo points to a DSPDATARDYMSGINFO structure, which can be summarized as follows:

```
// comment

// buffer or flag number

// flag value

//
```

flag
add
ress

//
boa
rd
han
dle

For more information, see section 5.1.3, DSPDATARDYMSGINFO Structure.

C/C++ Example

Below is an example showing a WM_DSPENGINE_FLGRDY message being received:

```
case WM_DSPENGINE_FLGRDY:  // semaphore message sent by DSP engine

    DWORD dwCount = 4096;  // process 4096 floating-point values each frame

    HGLOBAL hBuf = GlobalAlloc(GMEM_INIT | GMEM_MOVEABLE,
    dwCount*DS_GM_SIZE32);

    (float far*) pBuf = (float far*)GlobalLock(hBuf);  // lock memory before use

    // transfer below illustrates two common type of DSP memory architectures

    if (uMemArch == DS_GMA_VECTOR) {  // vector data memory

        uStatus = DSGetMem(hBoard, DS_GM_VECTOR_DATA_X,
        dwResultDataAddr, DS_GM_SIZE32, pBuf, dwCount);
    }
    else { // linear data memory

        uStatus = DSGetMem(hBoard, DS_GM_LINEAR_DATA_RT,
        dwResultDataAddr, DS_GM_SIZE32, pBuf, dwCount);
    }

    GlobalUnlock(hBuf);  // let memory go

    if (!uStatus) {
```

```
    // error occurred in DSGetMem call
    MessageBox(NULL, "DSGetMem problem", "Example", MB_OK);
}
else {

    // wait for semaphore again...
    DSWaitForFlag(hBoard, 0, NULL, DS_WFB_POLLED);
}
break;
```

A great deal of initialization is necessary before buffer ready and semaphore messages can be processed. The example above leaves out much of this; it is strongly recommended that the dscope.cpp example source code file (included with DSPower-HWLib software) be examined as at least one good illustration before implementing code similar to the above example. For example, uMemArch is assumed to have been initialized beforehand using the DSGetMemArch call.

In the above example, note that hBuf is created via a GlobalAlloc call, using a sizeof parameter equal to the total element count∗memory width per element. Memory pointers passed to DSGetMem and DSPutMem calls *must not point to memory segments less than the product of the dwCount and DS_GM_SIZExx parameters* used in the corresponding DSGetMem or DSPutMem calls. The resulting product is in bytes.

See Also

WM_DSPENGINE_BUFRDY

DSWaitForBuffer

DSWaitForFlag

DSRegisterEngineMsgWnd

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

WM_DSPENGINE_FUNCTIONERROR

WM_DSPENGINE_FUNCTIONERROR

hEngine = (HENGINE) wParam; // engine identifier

pErrorMsgInfo = (PERRORMSGINFO) lParam; // pointer to error information

The WM_DSPENGINE_FUNCTIONERROR engine status message is sent by Engine Manager after receiving an error message from a DSP Engine for which a function, display, or instrument has encountered an input parameters or other setup error. This error message is sent before the function starts.

Parameter	Description
hEngine	handle of DSP Engine encountering function error
pErrorMsgInfo	pointer to PERRORMSGINFO structure, which contains additional error information

Returns

The calling application does not need to return a specific value.

Comments

The pErrorMsgInfo parameter points to an ERRORMSGINFO structure, which can be summarized as follows:

```
// comment

// error number or type

// error/status string or error log filename
```

If nErrorNum is positive, then it represents a DSP Engine function initialization error number (as defined in DSP Engine documentation) and szStatusString is an error condition string. If the nErrorNum parameter is negative, then szStatusString is a pathname for an error log file. Currently, for

DSPower series software-compatible engines, error log files are only produced during compilation (i.e. currently, negative nErrorNum values only apply to WM_DSPENGINE_COMPILEERROR messages). For more information, see section 5.1.4, ERRORMSGINFO Structure.

C/C++ Example

Below is a partial source code example showing a WM_DSPENGINE_COMPILEERROR message being received:

```
char szErrorStatus[256];
char szLogFile[256];
HFILE hFile;
OFSTRUCT ofstruct;

case WM_DSPENGINE_FUNCTIONERROR:

    nErrorNum = (PERRORMSGINFO)lParam->nErrorNum;
    if (nErrorNum >= 0) {

        _fstrcpy(szErrorStatus, (PERRORMSGINFO)lParam->szErrorStatus);

        _fstrcat(szErrorStatus, "; Error Number = ");
        char tmpstr[30];
        itoa(nErrorNum, tmpstr, 10);
        _fstrcat(szErrorStatus, tmpstr);

        MessageBox(NULL, szErrorStatus, "Function Error Message", MB_OK);
    }
    else {

        _fstrcpy(szLogFile, PERRORMSGINFO)lParam->szErrorStatus);
        hFile = OpenFile(szLogFile, &ofstruct);

        // process error log file...
    }

    break;
```

Note if multiple engines are open, the wParam parameter can be used

to identify which engine has sent the error message.

See Also

WM_DSPENGINE_CODEGENERROR

WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_RUNTIMEERROR

DSRegisterEngineMsgWnd

DSExecEngineFunction

Related Users Guide Topics

Status Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_LEVEL

WM_DSPENGINE_LEVEL

```
hEngine = (HENGINE) wParam;     // engine identifier
dwLevel = (DWORD) lParam;       // maximum amplitude level
```

The WM_DSPENGINE_LEVEL engine update message is sent by Engine Manager when a DSP Engine amplitude level message is received during a high-level data acquisition, generation, or instrument function. The lParam parameter contains the maximum absolute amplitude found in each buffer of acquired or generated data.

Parameter	Description
hEngine	handle of engine sending level message

dwLevel	maximum amplitude level found in each buffer during high-level data acquisition, generation or instrument function

Returns

The calling application does not need to return a specific value.

Comments

The dwLevel parameter indicates the maximum absolute value found in each buffer of data processed during the high-level data acquisition, generation, or instrument function. Values range from 0 to the maximum full-scale value of the A/D or D/A converter(s) being used in the high-level function.

Note that values are always positive.

C/C++ Example

Below is an example of processing a WM_DSPENGINE_LEVEL message:

```
case WM_DSPENGINE_LEVEL:

    UpdateMeter(hMeter, lParam);  // lParam has maximum amplitude
    break;
```

See Also

WM_DSPENGINE_BUFNUM

WM_DSPENGINE_BUFSIZE

WM_DSPENGINE_SAMPFREQ

DSAcquireWvfrmFile

DSGenerateWvfrmFile

DSExecEngineFunction

Related Users Guide Topics

Update Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 4.5 Acquiring Data Using DSPower-HWLib Software

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_PROGSTATUS

WM_DSPENGINE_PROGSTATUS

```
hEngine = (HENGINE) wParam;          // engine identifier
wProgramStatus = LOWORD(lParam);     // program status
wFunctionID = HIWORD(lParam);        // function ID
```

The WM_DSPENGINE_PROGSTATUS program status message is sent by Engine Manager after receiving a program status message from a DSP Engine containing status information regarding an executing program.

Parameter	Description
hEngine	handle of engine for which program status information applies
wProgramStatus	program status, as defined by one of the values below:

Value	Meaning
DS_PS_FUNCDONE	engine function is complete
DS_PS_PROGDONE	program is complete

wFunctionID	identifier of function, if wProgramStatus above indicates a function has completed

Returns

The calling application does not need to return a specific value.

Comments

Depending on its flow of execution, a DSP Engine program may cause several DSP Engine functions to execute. Processing WM_DSPENGINE_PROGSTATUS functions provides a method to dynamically monitor the status of DSP Engine programs, regardless of the execution variations which are data-dependent or user-dependent.

Note that function ID values must be encoded into the source code generated and submitted to the DSP Engine to be executed as a native language program. This is done using the SetFunctionID DSP Engine function, which sets the current function ID for use in DSP Engine communication with the outside world.

Note that WM_DSPENGINE_PROGSTATUS program status callback messages are sent before any library functions, such as DSSendEngineCommand or DSExecEngineFunction,. which might be waiting for "program done" notification, are released. This situation can occur if these functions were executed by the calling application in asynchronous mode (uMode parameter == DS_EEF_ASYNC).

C/C++ Example

An example of processing a WM_DSPENGINE_PROGSTATUS message is shown below:

```
case WM_DSPENGINE_PROGSTATUS:

    if (LOWORD(lParam) == DS_PS_PROGDONE) {

        UpdateStatusBar(hProgStatus, "Engine Program Complete");
    }
    break;
```

See Also

DSRegisterEngineMsgWnd

DSExecEngineFunction

WM_DSPENGINE_ENGINESTATE

Related Users Guide Topics

Program Status Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_RUNTIMEERROR

WM_DSPENGINE_RUNTIMEERROR

```
hEngine = (HENGINE) wParam;          // engine identifier
pErrorMsgInfo =
(PERRORMSGINFO) lParam;              // pointer to error information
```

The WM_DSPENGINE_RUNTIMEERROR engine status message is sent by Engine Manager after receiving an error message from a DSP Engine which has encountered an error while executing a program.

Parameter	Description
hEngine	handle of DSP Engine encountering compiler error
pErrorMsgInfo	pointer to PERRORMSGINFO structure, which contains additional error information

Returns

The calling application does not need to return a specific value.

Comments

pErrorMsgInfo points to an ERRORMSGINFO structure, which can be summarized as follows:

// comment

// error number or type

// error/status string or error log filename

If nErrorNum is positive, then it represents a DSP Engine run-time error number (as defined in DSP Engine documentation) and szStatusString is an error condition string. If the nErrorNum parameter is negative, then szStatusString is a pathname for an error log file. Currently, for DSPower series software-compatible engines, error log files are only produced during compilation (i.e. currently, negative nErrorNum values only apply to WM_DSPENGINE_COMPILEERROR messages). For more information, see section 5.1.4, ERRORMSGINFO Structure.

C/C++ Example

Below is a partial source code example showing a WM_DSPENGINE_RUNTIMEERROR message being received:

```
char szErrorStatus[256], char tmpstr[30];

case WM_DSPENGINE_RUNTIMEERROR:

    _fstrcpy(szErrorStatus, (PERRORMSGINFO)lParam->szErrorStatus);
    _fstrcat(szErrorStatus, "; Error Number = ");
    itoa((PERRORMSGINFO)lParam->nErrorNum, tmpstr, 10);
    _fstrcat(szErrorStatus, tmpstr);

    MessageBox(NULL, szErrorStatus, "Run-Time Error Message", MB_OK);

    break;
```

Note if multiple engines are open, the wParam parameter can be used to identify which engine has sent the error message.

See Also

WM_DSPENGINE_CODEGENERROR

WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_FUNCTIONERROR

DSRegisterEngineMsgWnd

DSExecEngineFunction

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Status Messages from DSP Engines

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_SAMPFREQ

WM_DSPENGINE_SAMPFREQ

```
hEngine = (HENGINE) wParam;      // engine identifier
SampFreq = (float) lParam;       // sampling frequency
```

The WM_DSPENGINE_SAMPFREQ engine update message is sent by Engine Manager when a DSP Engine "actual sampling frequency" message is received during a high-level data acquisition, generation, or instrument function. The lParam parameter contains the actual sampling frequency being used during the function.

Parameter	Description
hEngine	handle of DSP Engine sending sampling frequency message
SampFreq	sampling frequency being used during high-level data acquisition, generation or instrument function

Returns

The calling application does not need to return a specific value.

Comments

The SampFreq parameter indicates the sampling frequency, in Hz, being used during the high-level data acquisition, generation, or instrument function. Sampling frequency values typically range from a few Hz up to 10 MHz or higher.

Although all high-level DSP Engine functions include a desired sampling rate parameter, the actual rate used may differ. This occurs because many types of DSP/analog hardware offer only a limited number of discrete sampling rate values. DSP Engine high-level functions then choose the physical value closest to the desired value, and report the choice using the WM_DSPENGINE_SAMPFREQ message.

C/C++ Example

Below is an example which shows a WM_DSPENGINE_SAMPFREQ message being processed:

```
float Fs;

case WM_DSPENGINE_SAMPFREQ:

    _fmemcpy(&Fs, &lParam, sizeof(float));  // store actual Fs being used
    char tmpstr[30];
    gcvt(Fs, 10, tmpstr);

    UpdateStatusBar(hSampFreq, tmpstr, "Sampling Frequency (in Hz)");
    break;
```

See Also

WM_DSPENGINE_BUFNUM

WM_DSPENGINE_BUFSIZE

WM_DSPENGINE_LEVEL

DSAcquireWvfrmFile

DSGenerateWvfrmFile

DSExecEngineFunction

Related Users Guide Topics

Update Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 4.5 Acquiring Data Using DSPower-HWLib Software

section 7.3 Executing Functions in DSP Engines

6.2 DSP Engine Messages

This section documents messages sent by DSP Engines. Normally, in DSPower series software, these messages are processed automatically, without user-intervention. For example, in response to a DSEngineOpen call, the DSPower-HWLib library receives and sends engine messages to Hypersignal-Macro series software in order to initialize the software in engine mode. From the user's point of view, in this case a C/C++ or MATLAB API, these messages are not visible--only the library call, in this case DSEngineOpen, is visible. Another example is the DSPower-Block Diagram package, which either processes DSP Engine messages directly or uses an intermediate layer such as the DSPower-HWLib

library. Again, from the user's point of view, engine messages are not visible--instead, an easy-to-use, graphical block diagram user-interface is visible.

For more information on the rationale and benefits of the DSP Engine concept, see Signalogic DSP Software System Guide section 1.1, Relationship of Signalogic DSP Software Components.

The actual numeric constants defined for DSP Engine messages are contained in the include files enmgr.h and engdef.h, which are supplied with DSPower-HWLib and DSPower-SPLib software, and which may be more current than this documentation.

DSP Engine Messages

DSPENG_ABORT

DSPENG_BUFRDY

DSPENG_CALCSAMPFREQ

DSPENG_CALCSAMPFREQRESULT

DSPENG_CANCELWAIT

DSPENG_CLIPRECT

DSPENG_CODEGENERROR

DSPENG_COMPILEERROR

DSPENG_DISABLEBOARD

DSPENG_DISPDIRFILESTEXT

DSPENG_DISPSTATUSTEXT

DSPENG_DISPWVFRMSUMTEXT

DSPENG_ERASERECT

DSPENG_ESC

DSPENG_EXIT

DSPENG_FLGRDY

DSPENG_FRAMERECT

DSPENG_FUNCTIONACTIVE

DSPENG_FUNCTIONDONE

DSPENG_FUNCTIONERROR

DSPENG_FUNCTIONPAUSE

DSPENG_GETBOARDCLASS

DSPENG_GETBOARDCLASSRESULT

DSPENG_GETHVARMEM

DSPENG_GETHVARMEMRESULT

DSPENG_GETMEM

DSPENG_GETMEMRESULT

DSPENG_GETMEMSIZE

DSPENG_GETMEMSIZERESULT

DSPENG_GETPARMS

DSPENG_HOLDBOARD

DSPENG_INITBOARD

DSPENG_INITBOARDSTATUS

DSPENG_LINEREL

DSPENG_LINETO

DSPENG_LOADFILE

DSPENG_LOADFILESTATUS

DSPENG_MOVEREL

DSPENG_MOVETO

DSPENG_PAINTRECT

DSPENG_PUTHVARMEM

DSPENG_PUTMEM

DSPENG_PUTMEMRESULT
DSPENG_RDBOARDREG
DSPENG_RDBOARDREGRESULT
DSPENG_REDRAW
DSPENG_REFRESH
DSPENG_REPORTBUFNUM
DSPENG_REPORTBUFSIZE
DSPENG_REPORTDISPID
DSPENG_REPORTFUNCID
DSPENG_REPORTLEVEL
DSPENG_REPORTSAMPFREQ
DSPENG_RESETBOARD
DSPENG_RUNBOARD
DSPENG_RUNEXTPROGASYNC
DSPENG_RUNEXTPROGSYNC
DSPENG_RUNTIMEERROR
DSPENG_SETAMP
DSPENG_SETBACKCOLOR
DSPENG_SETBITMAP
DSPENG_SETBOARDBASEADDR
DSPENG_SETBUFNUMID
DSPENG_SETCOLOR
DSPENG_SETCURTRACE
DSPENG_SETDISPID
DSPENG_SETDISPLAY
DSPENG_SETFUNCID
DSPENG_SETGCUR
DSPENG_SETLEVELID
DSPENG_SETOFS

DSPENG_SETPENCOLOR

DSPENG_SETPENMODE

DSPENG_SETRES

DSPENG_SETPIXEL

DSPENG_SETSTATISTICSID

DSPENG_SETTRACE

DSPENG_SETUNITS

DSPENG_SETUP

DSPENG_SETVOLTAGERANGE

DSPENG_SHUTDOWN

DSPENG_WAITFORBUF

DSPENG_WAITFORFLG

DSPENG_WAVEFORMEDITED

DSPENG_WRBOARDREG

WM_DSPENGINE_CMDDATA

DSPENG_ABORT

// command instructing engine to abort and unload from system, without saving currently open files or configuration

DSPENG_BUFRDY

// engine is indicating that buffer previously assigned by DSPENG_WAITFORBUF command is ready

DSPENG_CALCSAMPFREQ

// command instructing engine to calculate actual sampling frequency for specified DSP/analog hardware

DSPENG_CALCSAMPFREQRESULT

// engine is sending result of DSPENG_CALCSAMPFREQ command

DSPENG_CANCELWAIT

// command instructing engine to cancel pending wait requests

DSPENG_CLIPRECT

// engine is sending command to set clipping rectangle

DSPENG_CODEGENERROR

// engine encountered code generation error (WORD error number)

DSPENG_COMPILEERROR

// engine encountered compile error (WORD error number)

DSPENG_DISABLEBOARD

// command instructing engine to disable specified DSP/analog hardware

DSPENG_DISPDIRFILESTEXT

// engine is sending file directory status for display

DSPENG_DISPSTATUSTEXT

// engine is sending status text for display

DSPENG_DISPWVFRMSUMTEXT

// engine is sending waveform summary text for display

DSPENG_ERASERECT

// engine is sending command to erase rectangle

DSPENG_ESC

// command instructing engine to escape back up one level

DSPENG_EXIT

// command instructing engine to save open data and configuration files and unload from system

DSPENG_FLGRDY

// engine is indicating that flag previously assigned by DSPENG_WAITFORFLG command is ready

DSPENG_FRAMERECT

// engine is sending command to draw rectangle frame

DSPENG_FUNCTIONACTIVE

// engine function or program active

DSPENG_FUNCTIONDONE

// engine function or program complete

DSPENG_FUNCTIONERROR

// engine function or program terminated with error

DSPENG_FUNCTIONPAUSE

// engine function or program has paused

DSPENG_GETBOARDCLASS

// command instructing engine to return driver classification for specified DSP/analog hardware

DSPENG_GETBOARDCLASSRESULT

// engine is sending result of DSPENG_GETBOARDCLASS command

DSPENG_GETHVARMEM

// command instructing engine to read Hypersignal DSP Source Code Interface variable

DSPENG_GETHVARMEMRESULT

// engine is sending result of DSPENG_GETHVARMEM command

DSPENG_GETMEM

// command instructing engine to read data from DSP memory; for specified DSP/analog hardware

DSPENG_GETMEMRESULT

// engine is reporting completion status of previous DSPENG_GETMEM command

DSPENG_GETMEMSIZE

// command instructing engine to calculate maximum data memory size for specified DSP/analog hardware

DSPENG_GETMEMSIZERESULT

// engine is sending result of DSPENG_GETMEMSIZE size command

DSPENG_GETPARMS

// engine can accept parameters

DSPENG_HOLDBOARD

// command instructing engine to place specified DSP/analog hardware in hold mode

DSPENG_INITBOARD

// command instructing engine to initialize specified DSP/analog hardware

DSPENG_INITBOARDSTATUS

// engine is reporting result of DSPENG_INITBOARD command (DSP/analog hardware initialization status)

DSPENG_LINEREL

// engine is sending command to draw line relative

DSPENG_LINETO

// engine is sending command to draw line absolute

DSPENG_LOADFILE

// command instructing engine to download DSP Program File for specified DSP/analog hardware

DSPENG_LOADFILESTATUS

// engine is reporting completion status of DSPENG_LOADFILE command

DSPENG_MOVEREL

// engine is sending command to move graphics position relative

DSPENG_MOVETO

// engine is sending command to move graphics position absolute

DSPENG_PAINTRECT

// engine is sending command to paint rectangle

DSPENG_PUTHVARMEM

// command instructing engine to write Hypersignal DSP Source Code Interface variable

DSPENG_PUTMEM

// command instructing engine to write data to DSP memory for specified DSP/analog hardware

DSPENG_PUTMEMRESULT

// engine is reporting completion status of previous DSPENG_PUTMEM command

DSPENG_RDBOARDREG

// command instructing engine to read board register for specified DSP/analog hardware

DSPENG_RDBOARDREGRESULT

// engine is sending result of DSPENG_RDBOARDREG command

DSPENG_REDRAW

// command instructing engine to return to initial frame parameters

DSPENG_REFRESH

// command instructing engine to refresh display

DSPENG_REPORTBUFNUM

// engine is reporting current DSP buffer count for high-level function

DSPENG_REPORTBUFSIZE

// engine is reporting DSP buffer size being used in high-level function

DSPENG_REPORTDISPID

// engine is reporting new active display ID

DSPENG_REPORTFUNCID

// engine is reporting new active function ID

DSPENG_REPORTLEVEL

// engine is reporting current maximum absolute amplitude level for data buffer in high-level function

DSPENG_REPORTSAMPFREQ

// engine is reporting DSP/analog hardware sampling frequency being used in high-level function

DSPENG_RESETBOARD

// command instructing engine to reset specified DSP/analog hardware

DSPENG_RUNBOARD

// command instructing engine to run specified DSP/analog hardware

DSPENG_RUNEXTPROGASYNC

// run external program asynchronously with engine

DSPENG_RUNEXTPROGSYNC

// run an external program synchronously with engine

DSPENG_RUNTIMEERROR

// engine encountered run-time error (WORD error number)

DSPENG_SETAMP

// engine is sending command to set amplitude "hotspot" area

DSPENG_SETBACKCOLOR

// engine is sending command to set background color

DSPENG_SETBITMAP

// engine is sending command to set current bitmap

DSPENG_SETBOARDBASEADDR

// command instructing engine to set I/O and memory base addresses for specified DSP/analog hardware

DSPENG_SETBUFNUMID

// command instructing engine to set buffer number message ID

DSPENG_SETCOLOR

// command instructing engine to set current drawing color

DSPENG_SETCURTRACE

// engine requesting to set current display trace in multitrace display

DSPENG_SETDISPID

// command instructing engine to set current display ID

DSPENG_SETDISPLAY

// engine is sending command to set current display page

DSPENG_SETFUNCID

// command instructing engine to set current function ID

DSPENG_SETGCUR

// command instructing engine to set current graphics cursor

DSPENG_SETLEVELID

// command instructing engine to set amplitude level message ID

DSPENG_SETOFS

// command instructing engine to set current graphics offset

DSPENG_SETPENCOLOR

// engine is sending command to set pen color

DSPENG_SETPENMODE

// engine is sending command to set pen mode

DSPENG_SETRES

// command instructing engine to set current graphics resolution

DSPENG_SETPIXEL

// engine is sending command to set pixel

DSPENG_SETSTATISTICSID

// command instructing engine to set statistics message ID

DSPENG_SETTRACE

// engine is sending command to set current trace "hotspot" area

DSPENG_SETUNITS

// engine is sending command to set units "hotspot" area

DSPENG_SETUP

// engine sends this message after initializing and entering command mode

DSPENG_SETVOLTAGERANGE

// command instructing engine to set voltage ranges for specified DSP/analog hardware

DSPENG_SHUTDOWN

// engine sends this command as last action before shutdown

DSPENG_WAITFORBUF

// command instructing engine to assign wait-for-buffer request

DSPENG_WAITFORFLG

// command instructing engine to assign wait-for-flag request

DSPENG_WAVEFORMEDITED

// engine reports waveform has been edited

DSPENG_WRBOARDREG

// command instructing engine to write board register for specified DSP/analog hardware

WM_DSPENGINE_CMDDATA

WM_DSPENGINE_CMDDATA

```
wMsgType = wParam;                          // message type
pEngineSharedInfo =                         // pointer to shared memory
(PENGINESHAREDINFO) lParam;
```

The WM_DSPENGINE_CMDDATA message is sent by DSP Engines to external applications. Depending on the engine, it can be sent using a VxD or using direct Windows API SendMessage, PostMessage, or PostAppMessage calls.

Parameter	Description
wMsgType	message type, as defined below:

Value	Meaning
DS_CD_VXDMSG	message sent via VxD
DS_CD_APIMSG	message sent from standard API call

pEngineSharedInfo	pointer to shared memory, which can be accessed receiving application using ENGINESHAREDINFO

structure definition in engdef.h

Returns

The calling application does not need to return a specific value.

Comments

The WM_DSPENGINE_CMDDATA message contains a pointer to a shared memory area. Normally, this shared area contains bi-directional queues in which the external application and DSP Engine can exchange command and data information at maximum transfer rates. The bi-directional queues allow exchange of "command packets", which contain variable length data elements depending on the nature of the command. It is these commands which form many of the possible DSP Engine messages.

C/C++ Example

An example of processing a WM_DSPENGINE_CMDDATA message is shown below:

```
case WM_DSPENGINE_CMDDATA:

    pSharedEngineInfo = (PENGINESHAREDINFO)lParam;

    while (!IsQueueEmpty()) {

        Cmd = GetQueueByte();

        switch (Cmd) {

            // process commands
        }
    }
```

```
    break;
```

See Also

ENGINESHAREDINFO

DSExecEngineFunction

Related Users Guide Topics

section 7.3 Executing Functions in DSP Engines

7 Error and Status Conditions

Overview

The Error and Status Conditions section provides listings and technical descriptions of errors and status conditions, text displays, and reports which can occur in DSPower series and Hypersignal-Macro series software. Error and status conditions are further subdivided into the following categories:

- DSPower-Block Diagram
- DLL Functions
- DSP Engines

While DSPower-Block Diagram error and status conditions may cause display of an error number for documentation purposes, these conditions do not currently produce numeric constants which can be utilized by C/C++ or DSP Engine native language programs.

By contrast, DLL function error and status conditions are defined as numeric constants, which are returned by the functions themselves or by DSxxxGetErrorStatus routines located in various DLLs. They can also be accompanied by error and status text displays and reports which appear in the Engine Manager error/status listbox, if enabled (see Signalogic DSP Software Users Guide section 7.5.3, Error/Status Window in Engine Manager).

NOTE: In some extreme cases, a DSPower series DLL may also independently display its own message box, which indicates a serious system error, and which should be carefully noted and reported to Signalogic if it occurs.

DSP Engine error and status conditions and text displays are generated and returned by programs such as Hypersignal-Macro series software or MATLAB, even though

they may appear in status boxes or listboxes maintained by DSPower-Block Diagram software or a DSPower series DLL. For example, Engine Manager, which is part of DSPower-HWLib and DSPower-SPLib software, offers an error/status listbox which can be displayed as a debug tool, and which shows error and status conditions and text displays generated by DSP Engines as well as itself and other DSPower series DLLs. For more information, see Signalogic DSP Software Users Guide section 7.5.3, Error/Status Window in Engine Manager.

Error and Status Condition Categories

DSPower-Block Diagram Error and Status Conditions

DLL Error and Status Conditions

DSP Engine Error Conditions

See Also

Error and Status Condition Numeric List and Cross Reference

Below is a numeric listing of all error and status conditions, cross-referenced by page number.

Error and Status Condition Numeric List and Cross Reference

In the table below, there are two subtleties worth pointing out. First, positive error values indicate errors that could reasonably be expected to happen due to operator error, and

should be recoverable without difficulty. Negative error values indicate a more serious level of error, such as necessary program files not installed correctly, Windows resource problems, incorrect calling application usage of DLL functions, or internal DSPower series DLL errors.

Second, error values are grouped according to function. Error values -1 through -9 are general DLL errors (located in dllerr.h). Error values -10 through -49 can be generated by Engine Manager (enmgr.dll, located in enmgr.h), values -50 through -99 by Hardware Library (hwlib.dll, located in hwlib.h), and values -100 through -149 by Hardware Manager (hwmgr.dll, located in hwmgr.h).

Number	Error Definition	Page
	DSDSPFILENOTFOUND	
	DSHARDWARENOTRESPONDING	
	DSCOULDNOTOPENENGINE	
	DSINVALIDENGINEPATH	

DSCOU
LDNOT
CREAT
EBKGN
DWIND
OW

DSCOU
LDNOT
START
BKGND
TASK

DSNOB
KGND
WINDO
W

DSCOU
LDNOTI
NITIALI
ZEDLL

DSENGI
NETIM
EOUT

DSCOU
LDNOT
OPENPI
PE

DSCOU
LDNOT
START
ENGINE

DSNOE
NGINES
OPEN

DSNOE
NGINEF
OUNDI
NLIST

DSENGINEHANDLEWASNULL

DSENGINEMEMORYDISCARDED

DSINVALIDBOARDHANDLE

DSCOULDNOTOPENTEMPLATEFILE

DSCOULDNOTCREATESOURCEFILE

DSCREATINGENGINEPROGRAMERROR

DSUNABLETOALLOCATEMEMORY

DSUNABLETOREADTEMPLATEFILE

DSERRORINDATAPARAM

DSBOA
RDDOE
SNOTS
UPPOR
TCALL

DSINVA
LIDWO
RDLEN
GTH

DSINVA
LIDME
MORYT
YPE

DSINVA
LIDBOA
RDDESI
GNATO
R

DSBOA
RDNOT
INHWS
ETUPFI
LE

DSALL
BOARD
HANDL
ESALL
OCATE
D

DSINVA
LIDPRO
CESSO
RNUMB
ER

DSUNA
BLETO
READH
WSETU
PFILE

DSCOU
LDNOT
LOCKH
WLIST

DSBOA
RDDESI
GNATO
RNOTLI
STED

DSCOU
LDNOT
FINDH
WSETU
PFILE

DSCOU
LDNOTI
NCREA
SEHWLI
STMEM

DSHWS
ETUPFI
LENOT
YETRE
AD

Related Topics

General DLL Error and Status Conditions

Engine Error and Status Conditions

Hardware Manager Error and Status Conditions

Hardware Library Error and Status Conditions

Signal Processing Library Error and Status Conditions

General Topics

Error Handling in Engine Manager

Error Handling in Hardware Manager

Error/Status Window in Engine Manager

Error and Exception Handling in DSPower-HWLib and DSPower-SPLib Software

7.1 DSPower-Block Diagram Error and Status Conditions

This section covers error and status conditions and text displays generated by DSPower-Block Diagram software only. For error and status conditions and text displays generated by DSP Engines and shown in popup boxes or listboxes during interactive or block diagram operation, see section 7.3 below, DSP Engine Error and Status Conditions.

7.2 DLL Error and Status Conditions

This section defines the DLL error and status numerical constants which can be returned by functions in DSPower series DLLs.

In the DSPower series DLL error-handling scheme, a relatively small number of error and status constants are returned directly by the function called. However, most functions either return NULL, or zero, to indicate failure, and TRUE, or non-zero, to indicate success. Often times, the non-zero value returned on success is the actual return value of the function itself. Obvious examples include the DSEngineOpen and

DSAssignBoard calls in DSPower-HWLib software, which return engine and board handles, respectively.

Each DLL has a "base level" error/status routine which can be used to investigate error conditions. Currently, these calls are as follows:

Error/Status Call	DLL
	EngineManager(enmgr.dll)
	HardwareMa

nager(hwmgr.dll)

HardwareLibrary(hwlib.dll)

SignalPro

c
e
ss
in
g
L
ib
ra
ry
(s
pl
ib
.d
ll
)

In some cases, there may be additional "higher-level" error/status routines available. For example, Engine Manager also offers the DSGetEngineErrorStatus call to investigate problems associated with a specific engine; this call requires a valid engine handle as an input parameter.

Note also that error and status numerical constants are not the whole story. Engine Manager (enmgr.dll) maintains a "status listbox" which can be made visible using the DSShowEngMgrStatusWindow call. This status window shows a history of both status and error conditions resulting from most Engine Manager and Hardware Manager calls. For information about this status window, see Signalogic DSP Software Users Guide section 7.5.3, Error/Status Window in Engine Manager. For complete information on error

handling in Engine Manager, see Signalogic DSP Software Users Guide section 7.5, Error Handling in Engine Manager.

NOTE: In some extreme cases, a DSPower series DLL may also independently display its own message box, which indicates a serious system error, and which should be carefully noted and reported to Signalogic if it occurs.

DLL Error and Status Message Categories

7.2.1 General DLL Error and Status Conditions

Below is a source code example of using the Engine Manager "base level" error/status routine. The shows further investigation after a DSPower-HWLib software DSEngineOpen call failure:

```
if (!DSEngineOpen(szEnginePath, szWaveformPath)) {
    nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}
```

In the above example, the actual return value of DSEngineOpen would be NULL in the case of failure, since a non-NULL value would be a valid engine handle. Thus, to find out more information, the DSGetEngMgrErrorStatus call is used.

Below is a listing of ***general DLL*** error values which can be returned by DSGetEngMgrErrorStatus, along with a brief description of their meaning. These constants can be found in the dllerr.h include file; for a complete list of all DSPower series software error conditions by error number, see Error and Status Condition Numeric List and Cross-Reference, in the Overview section of this chapter, Error and Status Conditions.

Note that it is possible in some circumstances for the error values listed below to also be returned by the DSGetHWMgrErrorStatus, DSGetHWLibErrorStatus, and DSGetSPLibErrorStatus routines, besides DSGetEngMgrErrorStatus. Note also that in C/C++ programs, it is safest to treat these error definitions as integer types, since some values are "positive" and some "negative". See Error and Status Condition Numeric List and Cross Reference, above, for more information about this grouping.

General DLL Error Values

DSCOULDNOTCREATEBKGNDWINDOW

DSCOULDNOTSTARTBKGNDTASK

DSNOBKGNDWINDOW

DSCOULDNOTINITIALIZEDLL

Related Topics

General DLL Error and Status Conditions

Engine Error and Status Conditions

Hardware Manager Error and Status Conditions

Hardware Library Error and Status Conditions

Signal Processing Library Error and Status Conditions

General Topics

Error Handling in Engine Manager

Error Handling in Hardware Manager

Error/Status Window in Engine Manager

Error and Exception Handling in DSPower-HWLib and DSPower-SPLib Software

DSCOULDNOTCREATEBKGNDWINDOW

The background (hidden) window needed by enmgr.dll could not be created.

DSCOULDNOTSTARTBKGNDTASK

The background (hidden) task needed by enmgr.dll could not be started.

DSNOBKGNDWINDOW

The command failed because there was no background window.

DSCOULDNOTINITIALIZEDLL

Enmgr.dll itself could not be initialized.

7.2.2 Engine Manager Error and Status Conditions

Below is a source code example of using the Engine Manager "base level" error/status routine. The shows further investigation after a DSEngineOpen failure:

```
if (!DSEngineOpen(szEnginePath, szWaveformPath)) {
    nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}
```

In the above example, the actual return value of DSEngineOpen would be NULL in the case of failure, since a non-NULL value would be a valid engine handle. Thus, to find out more information, the DSGetEngMgrErrorStatus call is used.

Below is a listing of error values returned by DSGetEngMgrErrorStatus, along with a brief description of their meaning. These constants can be found in the enmgr.h include file; for a complete list of all DSPower series software error conditions by error number, see Error and Status Condition Numeric List and Cross-Reference, in the Overview section of this chapter, Error and Status Conditions. Note that in C/C++ programs, it is safest to treat these error definitions as integer types, since some values are "positive" and some "negative". See Error and Status Condition Numeric List

and Cross Reference, above, for more information about this grouping.

Engine Manager Error Values

DSCOULDNOTOPENENGINE

DSINVALIDENGINEPATH

DSENGINETIMEOUT

DSCOULDNOTOPENPIPE

DSCOULDNOTSTARTENGINE

DSNOENGINESOPEN

DSNOENGINEFOUNDINLIST

DSENGINEHANDLEWASNULL

DSENGINEMEMORYDISCARDED

Related Topics

General DLL Error and Status Conditions

Engine Error and Status Conditions

Hardware Manager Error and Status Conditions

Hardware Library Error and Status Conditions

Signal Processing Library Error and Status Conditions

General Topics

Error Handling in Engine Manager

Error Handling in Hardware Manager

Error/Status Window in Engine Manager

Error and Exception Handling in DSPower-HWLib and DSPower-SPLib Software

DSCOULDNOTOPEN ENGINE

Probably indicates DSP Engine executable file could not be located.

DSINVALIDENGINEP ATH

An invalid engine path was specified in a DSEngineOpen call.

DSENGINETIMEOUT

Time-out limits were exceeded in waiting for the DSP Engine to successfully open and initialize, or respond to a command or request.

DSCOULDNOTOPEN PIPE

The Pipe.386 VxD could not be opened for a DOS DSP Engine. Make sure that pipe.386 is listed in the "386Enh" section of the system.ini file on the x:\windows subdirectory.

DSCOULDNOTSTAR TENGINE

The DSP Engine could not be initialized correctly.

DSNOENGINESOPEN

A call was made requiring an engine handle but no engines were open.

DSNOENGINEFOUNDINLIST

A call was made using an engine handle which didn't specify an engine currently being maintained by enmgr.dll.

DSENGINEHANDLEWASNULL

A call was made using a NULL engine handle.

DSENGINEMEMORYDISCARDED

A call was made using an engine handle which pointed to memory already discarded.

7.2.3 Hardware Manager Error and Status Conditions

Below is a source code example of using the Hardware Manager "base level" error/status routine. The shows further investigation after a DSSetHWMgrCurVal failure:

```
if (!DSSetHWMgrCurVal(szBoardDesignator)) {

    nErrorType = DSGetHWMgrErrorStatus();
    ...do error processing...
}
```

In the above example, the actual return value of DSSetHWMgrCurVal would be NULL in the case of failure. Thus,

to find out more information, the DSGetHWMgrErrorStatus call is used.

Below is a listing of error values returned by DSGetHWMgrErrorStatus, along with a brief description of their meaning. These constants can be found in the hwmgr.h include file; for a complete list of all DSPower series software error conditions by error number, see Error and Status Condition Numeric List and Cross-Reference, in the Overview section of this chapter, Error and Status Conditions. Note that in C/C++ programs, it is safest to treat the above error definitions as integer types, since some values are "positive" and some "negative". See Error and Status Condition Numeric List and Cross Reference, above, for more information about this grouping.

Hardware Manager Error Values

DSUNABLETOREADHWSETUPFILE

DSCOULDNOTLOCKHWLIST

DSBOARDDESIGNATORNOTLISTED

DSCOULDNOTFINDHWSETUPFILE

DSCOULDNOTINCREASEHWLISTMEM

DSHWSETUPFILENOTYETREAD

Related Topics

General DLL Error and Status Conditions

Engine Error and Status Conditions

Hardware Manager Error and Status Conditions

Hardware Library Error and Status Conditions

Signal Processing Library Error and Status Conditions

General Topics

Error Handling in Engine Manager

Error Handling in Hardware Manager

Error/Status Window in Engine Manager

Error and Exception Handling in DSPower-HWLib and DSPower-SPLib Software

DSUNABLETOREADHWSETUPFILE

Unable to read the Hardware Setup File.

DSCOULDNOTLOCKHWLIST

Unable to lock the global memory hardware list maintained inside hwmgr.dll

DSBOARDDESIGNATORNOTLISTED

A board designator input to a hwmgr.dll function was not listed in the Hardware Setup File.

DSCOULDNOTFINDHWSETUPFILE

The Hardware Setup File could not be located.

DSCOULDNOTINCREASEHWLISTMEM

The global memory hardware list maintained inside hwmgr.dll could not be increased in size (to accommodate additional Hardware Setup File entries).

DSHWSETUPFILENOTYETREAD

A function in Hardware Manager was called which needed access to Hardware Setup File data, but the DSReadHWSetupFile function had not yet been called.

7.2.4 Hardware Library Error and Status Conditions

The DSPower-HWLib error handling scheme is straightforward. Most library calls return a non-zero value on success, and a zero (NULL) on failure. When calls into hwlib.dll fail, the DSGetHWLibErrorStatus and DSGetEngineErrorStatus calls can be used to attempt to learn more information about the most recent failure. These calls return error and status conditions associated with DSP/analog hardware and DSP/math functions.

Generally, at any given time both of these calls will return the same value. However, there is an

important difference: the latter retrieves error conditions associated with a particular engine, while the former returns an overall DLL value. Thus, if more than one engine is open at one time, it is possible that the DSGetHWLibErrorStatus value could be overwritten as a result of consecutive unrelated engine calls.

For error and status conditions associated with engine problems, the DSGetEngMgrErrorStatus and DSGetEngineErrorStatus routines (located in enmgr.dll) should be used. For error and status conditions associated with the Hardware Setup File or Hardware Selector, the DSGetHWMgrErrorStatus routine (located in hwmgr.dll) should be used.

NOTE: The DSGetEngineErrorStatus call is actually inside Engine Manager (enmgr.dll). Note also that Engine Manager maintains an "error/status listbox" window which can be made visible using the DSShowEngMgrStatusWindow call. This status window shows a history of both status and error conditions, including both those resulting from DSPower series DLL calls and generated by DSP Engines. For complete information on error handling in Engine Manager, see section 7.5, Error Handling in Engine Manager.

Below is an example showing error investigation after a DSAssignBoard failure:

```
if (!DSAssignBoard(hEngine, szBoardDesignator, uBusType, uIOBaseAddr,
uMemBaseAddr)) {

    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
```

In the above example, the actual return value of DSAssignBoard would be NULL in the case of failure, since a non-NULL value would be a valid board (hardware) handle. Thus, to find out more information, the DSGetEngineErrorStatus, DSGetEngMgrErrorStatus, and DSGetHWLibErrorStatus calls are checked. Note that DSGetEngineErrorStatus requires a valid engine handle as an input parameter. Thus, for problems related to a DSEngineOpen call, the DSGetEngineErrorStatus function should be avoided.

Below is a listing of error values returned by DSGetHWLibErrorStatus and DSGetEngineErrorStatus, along with a brief description of their meaning. These constants can be found in the hwlib.h include file; for a complete list of all DSPower series software error conditions by error number, see Error and Status Condition Numeric List and Cross-Reference, in the Overview section of this chapter, Error and Status Conditions. Note that in C/C++

programs, it is safest to treat these error definitions as integer types, since some values are "positive" and some "negative". See Error and Status Condition Numeric List and Cross Reference, above, for more information about this grouping.

Hardware Library Error Values

DSDSPFILENOTFOUND

DSHARDWARENOTRESPONDING

DSINVALIDBOARDHANDLE

DSCOULDNOTOPENTEMPLATEFILE

DSCOULDNOTCREATESOURCEFILE

DSCREATINGENGINEPROGRAMERROR

DSUNABLETOALLOCATEMEMORY

DSUNABLETOREADTEMPLATEFILE

DSERRORINDATAPARAM

DSBOARDDOESNOTSUPPORTCALL

DSINVALIDWORDLENGTH

DSINVALIDMEMORYTYPE

DSINVALIDBOARDDESIGNATOR

DSBOARDNOTINHWSETUPFILE

DSALLBOARDHANDLESALLOCATED

DSINVALIDPROCESSORNUMBER

Related Topics

General DLL Error and Status Conditions

Engine Error and Status Conditions

Hardware Manager Error and Status Conditions

Hardware Library Error and Status Conditions

General Topics

Error Handling in Engine Manager

Error Handling in Hardware Manager

Error/Status Window in Engine Manager

Error and Exception Handling in DSPower-HWLib and DSPower-SPLib Software

DSDSPFILENOTFOUND

Typically occurs after DSLoadFileBoard or DSLoadFileProcessor call. Indicates a DSP Program File could not be found.

DSHARDWARENOTRESPONDING

Typically occurs after a DSInitBoard call. Indicates DSP/analog hardware not present or not installed correctly.

DSINVALIDBOARDHANDLE

Board handle was invalid.

DSCOULDNOTOPENTEMPLATEFILE

A template file needed by a high-level function could not be opened.

DSCOULDNOTCREATESOURCEFILE

A DSP Engine source code file being generated by a high-level function could not be created.

DSCREATINGENGINEPROGRAMERROR

An error occurred when compiling or linking a DSP Engine source code file(s).

DSUNABLETOALLOCATEMEMORY

Memory for the function could not be allocated

DSUNABLETOREADTEMPLATEFILE

An error occurred reading a template file needed b a high-level function.

DSERRORINDATAPARAM

A parameter passed to a high level function in a data structure was invalid.

DSBOARDDOESNOTSUPPORTCALL

The hardware type does not support the call.

DSINVALIDWORDLENGTH

A wordlength parameter passed to a memory transfer function was invalid.

DSINVALIDMEMORYTYPE

A memory type parameter passed to a memory transfer function was invalid.

DSINVALIDBOARDDESIGNATOR

The board designator string was not recognized, or not listed in the Hardware Setup File.

DSBOARDNOTINHWSETUPFILE

The board handle or designator is no longer listed in the Hardware Setup File

DSALLBOARDHANDLESALLOCATED

No more board handles can be allocated.

DSINVALIDPROCESSORNUMBER

The processor number was out of range vs. the number of processors on the hardware.

7.2.5 Signal Processing Library Error and Status Conditions

This section not available yet.

7.3 DSP Engine Error and Status Conditions

DSP Engine error and status conditions and text displays are generated and returned by programs such as Hypersignal-Macro series software or MATLAB, even though they may appear in status boxes or listboxes maintained by DSPower-Block Diagram software or a DSPower series DLL. For example, Engine Manager, which is part of DSPower-HWLib and DSPower-SPLib software, offers an error/status listbox which can be displayed as a debug tool, and which shows error and status conditions and text displays generated by DSP Engines as well as itself and other DSPower series DLLs. For more information, see Signalogic DSP Software Users Guide section 7.5.3, Error/Status Window in Engine Manager.

7.3.1 Hypersignal Error and Status Conditions

This section not available yet.

7.3.2 MATLAB Error and Status Conditions

This section not available yet.